INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE
Departamento de Matemáticas y Física

Examen de Métodos de Inteligencias Artificial

Nombre:
-Ana Sofía Avila Gálvez 745247
-Ana Paula Moreno Haro 744069
-Samuel Gómez Jiménez 744347
-Sharon Nicole Astrid Manzano Aceves 748293

Fecha: 27/10/25

Instrucciones. *Lee cuidadosamente cada pregunta y contesta lo que se te pide. No olvides escribir subir todos los archivos .m elaborados para cada inciso que requiera de uso de software y pegar el código de programa indentado y comentado.*

Nota: *Limitarse al uso de los modelos analizados en el segundo bloque del curso.*

1. (**5 puntos**) ¿Con el modelo de Perceptrón simple, es posible realizar clasificación no binaria? Si es posible describa la metodología para llevarlo a cabo.

Sí se puede, usando varios perceptrones simples (uno por clase) con métodos como uno contra todos o uno contra uno. En este caso, cada perceptrón se entrena para reconocer una clase diferente y la muestra se clasifica según el perceptrón que dé la salida más alta.

2. (**5 puntos**) Elige aleatoriamente un número del 4 al 9. Si se tienen tantas variables de entrada en un problema de Regresión como dicho número aleatorio ¿Cuántos pesos requiere un polinomio de grado tres si cuenta con todas las combinaciones posibles con esa cantidad de variables?

Si el número aleatorio es 6, un polinomio de grado 3 requiere 84 pesos, considerando todas las combinaciones posibles de las 6 variables hasta el tercer grado. Esto se calcula con la fórmula de combinaciones con repetición (n + g sobre g), donde n es el número de variables y “g” el grado del polinomio. Esta fórmula cuenta todas las combinaciones posibles de términos lineales, cuadráticos y cúbicos que pueden formarse con esas variables. El resultado significa que el modelo necesita 84 coeficientes o pesos diferentes para representar todas las relaciones posibles entre las variables hasta el grado tres.

$$\text{Número de pesos} = \binom{n+g}{g} = \binom{6+3}{3} = 84$$

3. **(5 puntos)** Indica qué es una neurona muerta en una red competitiva.

Una neurona muerta es una que nunca gana ni se activa, es decir, no responde a ningún patrón de entrada. Esto puede ocurrir cuando su peso inicial queda muy alejado de los datos, por lo que no llega a ajustarse y al final no representa a ningún grupo dentro de la red.

4. **(5 puntos)** Explica dos formas de detectar el sobreajuste (overfitting) en una red competitiva.

Hay sobreajuste cuando el modelo tiene un error bajo durante el entrenamiento pero un error alto al probarlo con datos nuevos, lo que indica que memorizó (imitó) los ejemplos en lugar de generalizar. También puede detectarse cuando las neuronas solo aprenden casos muy específicos o muy pocos datos, dejando otras regiones del espacio sin representar correctamente.

Problemas

5. (25 puntos) El conjunto de datos de la NASA **(Archivo airfoil.txt)** comprende perfiles aerodinámicos NACA 0012 de diferentes tamaños a diversas velocidades de túnel de viento y ángulos de ataque. La envergadura del perfil aerodinámico y la posición del observador fueron las mismas en todos los experimentos.

x_1 : Frequency (measured in Hz)
x_2 : Attack-angle (measured in deg)
x_3 : Chord-length (measured in m)
x_4 : Free-stream-velocity (measured in m/s)
x_5 : Suction-side-displacement-thickness (measured in m)
y: Scaled-sound-pressure (measured in dB)

```
clear all; close all; clc;
load airfoil.txt
%% Train-Test
data = airfoil;
index = round(size(data,1) * 0.8);
X = normalize(data(:,1:5));
Xtrain = X(1:index,:);
Xtest = X(index+1:end, :);
X1 = Xtrain(:,1);
X2 = Xtrain(:,2);
X3 = Xtrain(:,3);
X4 = Xtrain(:,4);
X5 = Xtrain(:,5);
Y = data(:,6);
Ytrain = Y(1:index,:);
Ytest = Y(index+1:end, :);
%% Número de datos
n = size(Xtrain,1);
%% Modelo a)
Xa = [ones(n,1), X1, X3, X5, X2.^2, X5.*X1, X1.*X3, (X4.^2).*X2, X4.^2,
X2.^3];
Wmc_a = inv(Xa' * Xa) * Xa' * Ytrain;
imprimir_coeficientes(Wmc_a)
Yg = Xa * Wmc_a; % Y estimada
E = Ytrain - Yg; % Error
J_a = (E' * E) / (2*n) % Costo
%% Modelo b)
[Xa coef] = func_polinomio2(Xtrain, 2);
Wmc_b = inv(Xa' * Xa) * Xa' * Ytrain;
decod_func_polinomio2(Xa, coef, Wmc_b)
Yg = Xa * Wmc_b; % Y estimada
E = Ytrain - Yg; % Error
```

Examen de Métodos de Inteligencias Artificial

```
J_b = (E' * E) / (2*n) % Costo
%% Estimaciones
new_data = [1000 1.5 0.2286 39.6 0.00229336;
            6300 0 0.1524 55.5 0.00253511;
            315 2.7 0.0254 71.3 0.00372371;
            12500 5.4 0.1524 31.7 0.111706];
new_data = normalize(new_data);
Xa_new_data = func_polinomio2(new_data, 2);
Yg_new_data = Xa_new_data * Wmc_b
```

a) (7.5 puntos) Determine los coeficientes de un modelo de que relacione *y* con *x*1, *x*2 y x4 de la siguiente forma: $y=b_0 + b_1x_1+ b_2x_3 + b_3x_5 + b_4x_2^2 + b_5x_5x_1 + b_6x_1x_3 + b_7x_4^2x_2+b_8x_4^2+ b_9x_2^3$

```
Coeficientes del modelo de regresión:
b0 = 122.803445
b1 = -4.831469
b2 = -2.258830
b3 = -3.749185
b4 = 0.416014
b5 = -3.047141
b6 = -1.525224
b7 = 0.127378
b8 = 0.928025
b9 = -0.611489
```

```
J_a =

    9.1507
```

b) (7.5 puntos) Determine los coeficientes de un modelo de segundo orden que contemple todas las combinaciones de las variables que considere pertinentes.

```
y_hat = +124.437*(1) -0.431*(x5^1) +1.458*(x4^1) -3.982*(x3^1) -3.240*(x2^1) -5.051*(x1^1)
+0.204*(x4^1 * x5^1) -2.238*(x3^1 * x5^1) +0.299*(x3^1 * x4^1) -0.424*(x2^1 * x5^1)
+0.145*(x2^1 * x4^1) +1.268*(x2^1 * x3^1) -2.248*(x1^1 * x5^1) +0.237*(x1^1 * x4^1)
-1.615*(x1^1 * x3^1) -0.686*(x1^1 * x2^1) -0.553*(x5^2) +0.078*(x4^2) +0.999*(x3^2)
-0.657*(x2^2) -0.068*(x1^2)
```

```
J_b =

    7.9295
```

c) (5 puntos) Indique cuál de los dos modelos anteriores es mejor y por qué (en qué parámetro se basó).

El modelo b es preferible al modelo a porque obtiene un costo menor. Dado que J mide el error cuadrático promedio de las predicciones, un valor más bajo implica que el modelo b predice Y con menor error. Esto indica que el modelo polinomial de segundo orden, que incluye todas las interacciones y términos cuadráticos, captura mejor la relación entre las variables y la salida que el modelo a, que es más restringido en su forma tan específica.

d) (5 puntos) Según el modelo que haya sido mejor determine la presión del sonido escalada (y) para los siguientes casos:

x_1	x_2	x_3	x_4	x_5
1000	1.5	0.2286	39.6	0.00229336
6300	0	0.1524	55.5	0.00253511
315	2.7	0.0254	71.3	0.00372371
12500	5.4	0.1524	31.7	0.0111706

```
Yg_new_data =

  126.1691
  126.0341
  132.2366
  100.4645
```

6. (**30 puntos**) Para el set de datos contenidos en el archivo gato.txt realizar los siguiente:

```
clear all; close all; clc;
T = readtable('gato.xlsx');
dataTable = T;
% X = jugadas (9 columnas), Y = resultado (columna 10)
X = dataTable(:, 1:9);
Y = dataTable(:, 10);
% Convertir las jugadas a números: x=1, o=-1, b=0
X = table2array(varfun(@(x) double(strcmp(x,'x')) - double(strcmp(x,'o')),
X));
% Convertir la salida: "positive" = 1 (ganadora), "negative" = 0
Y = strcmp(table2array(Y), 'positive');
Y = double(Y(:));
%% Train-Test
cv = cvpartition(Y,'holdout',0.1);
% Datos de entrenamiento.
Xtrain = X(training(cv),:);
Ytrain = Y(training(cv));
% Datos de prueba.
Xtest = X(test(cv),:);
Ytest = Y(test(cv));
%% Escalamiento
Xtrain = normalize(Xtrain);
Xtest = normalize(Xtest);
%% Modelado
n = size(Xtrain,1);  % Cantidad de datos
grado = 3;
[Xa coef] = func_polinomio2(Xtrain,grado);  % La forma del modelo
% Inicialización de parámetros
W = zeros(size(Xa, 2), 1);  % Pesos iniciales
[J, dJdW] = func_costo(W, Xa, Ytrain);
options = optimset('GradObj', 'on', 'MaxIter', 1000);
[Wopt, Jopt] = fminunc(@(W)func_costo(W, Xa, Ytrain), W, options);
%% Train Performance
Vtrain = Xa * Wopt;
Ygtrain = round(1./(1+exp(-Vtrain)));
% Matriz de Confusión
A_train = confusionmat(Ytrain, Ygtrain);
figure(1)
confusionchart(A_train,[0 1])
%confusionchart(A,[0 1])
TPtrain = A_train(2,2);
TNtrain = A_train(1,1);
FPtrain = A_train(1,2);
FNtrain = A_train(2,1);
exa_train = (TPtrain+TNtrain) / (TPtrain+TNtrain+FPtrain+FNtrain);  %
Exactitud
pre_train = TPtrain / (TPtrain+FPtrain);  % Precisión
rec_train = TPtrain / (TPtrain+FNtrain);  % Recall
```

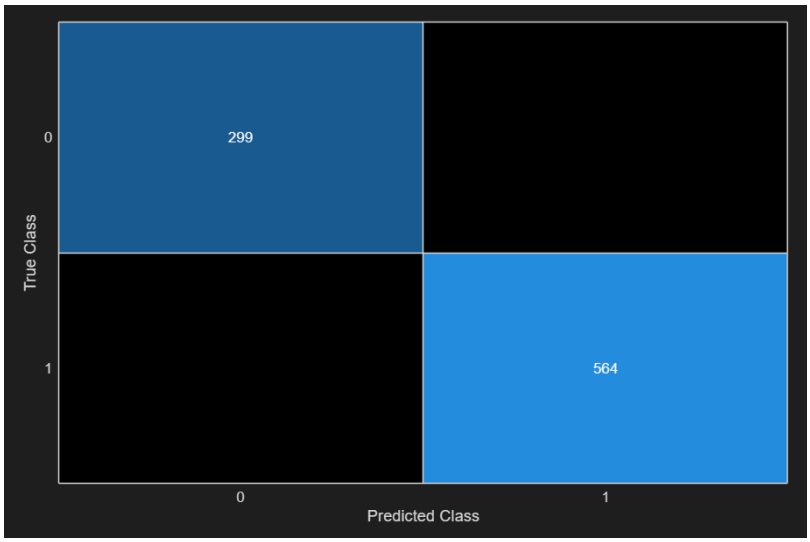
Examen de Métodos de Inteligencias Artificial

```
fprintf('\n--- ENTRENAMIENTO ---\n');
fprintf('Exactitud: %.2f\nPrecisión: %.2f\nRecall: %.2f\n', exa_train,
pre_train, rec_train);
%% Test Performance
Xatest = func_polinomio2(Xtest, grado);
Vtest = Xatest * Wopt;
Ygtest = round(1./(1+exp(-Vtest)));
% Matriz de Confusión
A_test = confusionmat(Ytest, Ygtest);
figure(2)
confusionchart(A_test,[0 1])
%confusionchart(A,[0 1])
TPtest = A_test(2,2);
TNtest = A_test(1,1);
FPtest = A_test(1,2);
FNtest = A_test(2,1);
exa_test = (TPtest+TNtest) / (TPtest+TNtest+FPtest+FNtest);  % Exactitud
pre_test = TPtest / (TPtest+FPtest);  % Precisión
rec_test = TPtest / (TPtest+FNtest);  % Recall
fprintf('\n--- TEST ---\n');
fprintf('Exactitud: %.2f\nPrecisión: %.2f\nRecall: %.2f\n', exa_test,
pre_test, rec_test);
%% Predicciones nuevas
Xpred = [
   0  1  1  0  1  0 -1 -1 -1;
   0  1 -1  1  1  0  1  0 -1;
   1  0  1  0  1  0 -1  1  0;
   0  1  0  1  0  1  1  0  1;
   1  0  1  0  1  0  0  1  1
];
Xpred = normalize(Xpred);
Xapred = func_polinomio2(Xpred, grado);
Vpred = Xapred * Wopt;
Ygpred = round(1./(1+exp(-Vpred)));
disp('--- PREDICCIONES NUEVAS ---');
disp('1 = Ganadora, 0 = No ganadora');
disp(Ygpred);
```

a) **(10 puntos)** Convertir la información de la base de datos para trabajarla (Si se hace desde Excel adjuntar el archivo a la entrega).

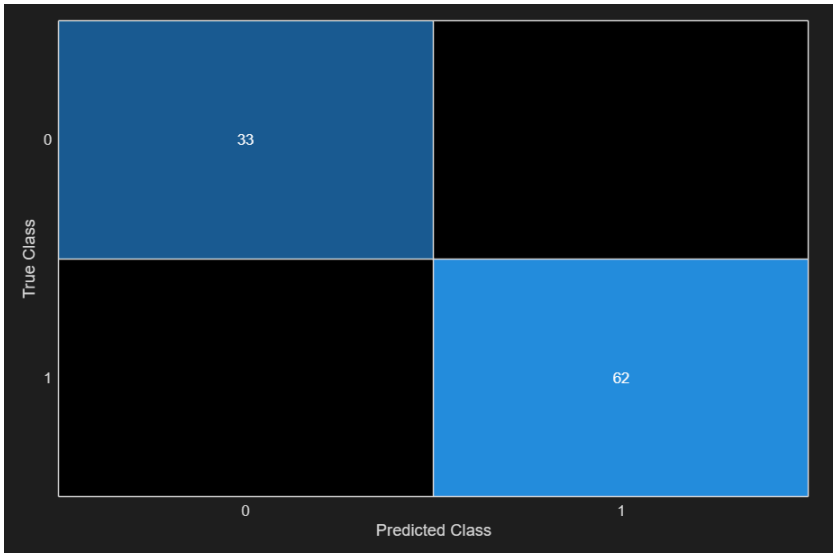
b) **(5 puntos)** Obtener un modelo de clasificación usando el 90% de los datos. Un modelo aceptable debe tener medidas de desempeño (Exactitud, precisión y recall superiores al 75%).

c) **(5 puntos)** Proporcione la matriz de confusión y medidas de desempeño del entrenamiento.




```
--- ENTRENAMIENTO ---
Exactitud: 1.00
Precisión: 1.00
Recall: 1.00
```

d) (5 puntos) Proporcionar la matriz de confusión y las medidas de desempeño de la validación.




```
--- TEST ---
Exactitud: 1.00
Precisión: 1.00
Recall: 1.00
```

e) (5 puntos) Usando el modelo indique si las siguientes jugadas son o no ganadoras para el jugador que usa x.

o,x,x,b,x,x,o,o,o
b,x,o,x,x,o,x,o,o
x,o,x,o,x,b,x,b,o
o,x,b,b,o,b,x,x,o
x,b,o,x,o,b,o,b,x

La descripción de los datos se puede consultar en:

https://archive.ics.uci.edu/dataset/101/tic+tac+toe+endgame

```
--- PREDICCIONES NUEVAS ---
1 = Ganadora, 0 = No ganadora
     0
     0
     0
     1
     1
```

7. **(25 puntos)** En el archivo **base_semillas.xlsx** contiene información de una muestra de 210 semillas de trigo.
   La visualización de alta calidad de la estructura interna del kernel se detectó mediante una técnica de rayos X blandos. Las imágenes se registraron en placas KODAK de rayos X de 13x18 cm. Los estudios se llevaron a cabo utilizando grano de trigo cosechado en una cosechadora proveniente de campos experimentales, explorado en el Instituto de Agrofísica de la Academia de Ciencias de Polonia en Lublin.

```
clear all;

close all;

clc;

rng(7);

%% Cargar Datos

load base_semillas.mat

data = base_semillas; % copia de los datos

[d, N] = size(data); % obtiene el número de filas (d) y columnas (N) de
la matriz data

% Estandarización de los datos

% para que todas las variables tengan igual peso en las distancias

mu = mean(data, 2); % media de cada variable (por fila)

sd = std(data, 0, 2); % desviación estándar de cada variable
```

```
sd(sd == 0) = 1; % evita división por cero si alguna sd es 0

data = (data - mu) ./ sd; % resta la media y divide por la desviación
estándar

%% Parámetros generales

epochs = 200; % número de épocas

eta = 0.05; % velocidad de convergencia

neuronas_lista = [2, 3]; % 2 y 3 neuronas

% Función de costo J: promedio de distancia Euclídea al centro asignado

% Métrica de error (costo promedio de distancias a su centro)

calcJ = @(data, W, gan) mean( sqrt( sum( (data - W(:,gan)).^2 , 1 ) ) );

% Se usó la medición de la distancia Euclídea por ser la más exacta

% guardar resultados

RESULT = struct([]);

%% Entrenamiento competitivo

for caso = 1:length(neuronas_lista)

    nn = neuronas_lista(caso); % número de neuronas que se van a probar
(2 o 3)

   % Elegir centros iniciales al azar (a partir de muestras reales)

   idx0 = randperm(N, nn);

   W = data(:, idx0); % cada columna de W es una neurona o centro

   % Entrenamiento de la red

   for nepoc = 1:epochs % repetir varias veces sobre todos los datos

       for k = 1:N % recorrer cada muestra del conjunto

           xk = data(:,k); % tomar una muestra

           % Calcular qué tan lejos está la muestra de cada neurona

           for j = 1:nn

                   distancia(1,j) = sum((xk - W(:,j)).^2); % distancia
Euclídea al cuadrado

           end

           % Buscar la neurona más cercana (ganadora)

           [~, ind] = min(distancia);

           % Mover el peso de la neurona ganadora hacia la muestra
```

Examen de Métodos de Inteligencias Artificial

```
            W(:,ind) = W(:,ind) + eta*(xk - W(:,ind));

        end

    end

    % Asignación final de cada muestra al centro más cercano

    ganador = zeros(1,N); % vector donde se guarda qué neurona ganó para
cada muestra

    for k = 1:N  % recorre todas las muestras

        xk = data(:,k); % tomar la muestra k

        for j = 1:nn % calcular la distancia a cada neurona

             distancia(1,j) = sum((xk - W(:,j)).^2); % distancia Euclídea
al cuadrado

        end

         [~, ganador(k)] = min(distancia); % guardar la neurona que quedó
más cerca (ganadora)

    end

    % Conteo por grupo (INCISOS: a y c)

    conteos = zeros(1,nn); % Un vector para contar cuántas muestras tiene
cada grupo

    for j = 1:nn

         conteos(j) = sum(ganador == j); % cuenta cuántas veces ganó cada
neurona

    end

    % Costo J (INCISOS: b y d)

    J = calcJ(data, W, ganador); % calcula el costo promedio (distancia
media a su centro asignado)

    % Guardar resultados

    RESULT(caso).nn = nn; % número de neuronas del modelo

    RESULT(caso).W = W; % pesos finales (centros)

      RESULT(caso).ganador = ganador; % índice del grupo ganador por
muestra

    RESULT(caso).conteos = conteos; % cantidad de muestras por grupo

    RESULT(caso).J = J;  % costo del modelo (promedio de distancias)

end
```

a) (5 puntos) Use un modelo con 2 neuronas e indique cuántos datos hay en cada grupo.

Examen de Métodos de Inteligencias Artificial

```
(a) Modelo con 2 neuronas:
Conteos por grupo: 117 93
```

Esto indica que al usar dos neuronas, la red dividió las 210 semillas en dos conjuntos, el primer grupo con 117 semillas y el segundo con 93. En este caso, la red encontró que con dos centros es posible separar los datos en dos agrupaciones principales.

b) **(5 puntos)** Calcule el costo del modelo anterior.

```
(b) Costo J (2 neuronas): 1.725481
```

Este valor representa la distancia promedio entre cada semilla y el centro del grupo al que pertenece. Un valor de 1.72 significa que en promedio las semillas están a esa distancia (en unidades normalizadas) de su centro.

c) **(5 puntos)** Use un modelo con 3 neuronas e indique cuántos datos hay en cada grupo.

```
(c) Modelo con 3 neuronas:
Conteos por grupo: 63 71 76
```

Aquí la red competitiva formó tres grupos con un tamaño relativamente equilibrado. Eso indica que los datos pueden distribuirse bien en tres regiones distintas dentro del espacio de variables.

d) **(5 puntos)** Calcule el costo del modelo anterior.

```
Costo J (3 neuronas): 1.358058
```

Esto significa que las semillas quedaron más cerca de sus centros, por lo que los grupos formados son más pequeños, claros y mejor definidos que en el modelo anterior.

e) **(5 puntos)** Indique qué modelo es mejor, inciso a) o c), justificando plenamente su respuesta.

El mejor modelo es el de 3 neuronas porque tiene menor costo J (1.358058), los grupos son más precisos y reflejan mejor las diferencias entre las semillas.

f) **(10 puntos)** Presente un código de Matlab para calcular J.

La fórmula de J (**J = mean(sqrt(sum((data - W(:,gan)).^2 , 1)))**) calcula el costo promedio del modelo. Lo que hace es medir qué tan lejos está cada muestra del centro o neurona que le corresponde. Primero se obtiene la distancia euclídea entre cada muestra y su centro, luego se suman todas esas distancias y se saca el promedio. El valor de J representa qué tan compactos están los grupos; si J es pequeño, las muestras están más cerca de su centro y el modelo es mejor.