Práctica 1: Implementación del perceptrón multicapa

Convocatoria de enero (curso académico 2023/2024)

Asignatura: Introducción a los modelos computacionales
4º Grado Ingeniería Informática (Universidad de Córdoba)

27 de septiembre de 2023

Resumen

Esta práctica sirve para familiarizar al alumno con los modelos computacionales de redes neuronales, en concreto, con el perceptrón multicapa. Para ello, el alumno deberá implementar el algoritmo de retropropagación básico para el perceptrón multicapa y comprobar el efecto de distintos parámetros (arquitectura de la red, factor de momento, etc.). La entrega se hará utilizando la tarea en Moodle habilitada al efecto. Se deberán subir en un único fichero comprimido todos los entregables indicados en este guión. El día tope para la entrega es el **17 de octubre**. En caso de que dos alumnos entreguen prácticas copiadas, no se puntuarán ninguna de las dos.

1. Introducción

El trabajo que se va a realizar en la práctica consiste en implementar el algoritmo de retropropagación para entrenar un perceptrón multicapa para un problema concreto.

Para ello, se desarrollará un programa capaz de realizar este entrenamiento, con distintas posibilidades en cuanto a la parametrización del mismo. Este programa se utilizará para entrenar modelos que predigan, de la forma más correcta posible, la(s) variable(s) objetivo(s) de un conjunto de bases de datos disponible en Moodle y se realizará un análisis de los resultados obtenidos. **Este análisis influirá en gran medida en la calificación de la práctica**.

En el enunciado de esta práctica, se proporcionan valores orientativos para todos los parámetros del algoritmo. Sin embargo, se valorará positivamente si el alumno encuentra otros valores para estos parámetros que le ayuden a mejorar los resultados obtenidos. La única condición es que no se puede modificar el número máximo de iteraciones totales (1000 del bucle externo y N del bucle interno, por lo tanto, en todo caso está limitado a $1000 \cdot N$, donde N es el número de patrones de la base de datos) y que los valores de los parámetros deben ser constantes para todas las bases de datos o, en todo caso, depender del citado tamaño.

La sección 2 describe una serie de pautas generales a la hora de implementar el algoritmo de retropropagación. La sección 3 explica los experimentos que se van a realizar una vez implementado el algoritmo. Finalmente, la sección 4 especifica los ficheros que se deben entregar para esta práctica.

2. Implementación del algoritmo de retropropagación

Se deben de seguir las indicaciones aportadas en las diapositivas de clase, donde se proporciona un pseudocódigo orientativo y la estrategia general para realizar la implementación del algoritmo. Algunas características que deben ser aclaradas son las siguientes:

- *Arquitectura de la red*: Pretendemos desarrollar un algoritmo genérico, donde la estructura del perceptrón multicapa sea libre y a escoger por el usuario. El número de capas H cumplirá $H \geq 2$, es decir, como mínimo habrá una capa de entrada (capa 0), una capa oculta

(capa 1) y una capa de salida (capa 2), pero el número de capas ocultas puede ser mayor. Además, el usuario podrá especificar el número de neuronas de cada una de las capas ocultas (el número de neuronas de entrada y de salida vienen determinadas por el problema a resolver). Se tomará el mismo número de neuronas para todas las capas ocultas.

- *Tipología de las neuronas*: Todas las neuronas, salvo las de la capa de entrada, serán de tipo sigmoide y dispondrán de sesgo. Su expresión será, por tanto:

$$out_j^h = \frac{1}{1 + \exp(-w_{j0}^h - \sum_{i=1}^{n_{h-1}} w_{ji}^h out_i^{h-1})}.$$

- *Actualización de los pesos*: utilizaremos una tasa de aprendizaje de $\eta = 0{,}1$ y un factor de momento de $\mu = 0{,}9$. Recuerda que el concepto de momento para mejorar la convergencia añade un nuevo término para que los cambios anteriores influyan en la dirección del cambio actual. Así, los pesos que empiezan a moverse en una determinada dirección, tienden a moverse en esa dirección. El parámetro mu (μ) controla el efecto del momento (último término de la siguiente ecuación, que se actualizará y guardará para estar disponible en la siguiente iteración).

$$w_{ji}^h = w_{ji}^h - \eta \Delta w_{ji}^h - \mu \left(\eta \Delta w_{ji}^h (t-1)\right)$$

- *Modo de funcionamiento*: En esta primera práctica, el algoritmo trabajará en modo *on-line* o en línea, es decir, por cada patrón de entrenamiento (bucle interno), calcularemos el error y modificaremos los pesos de acuerdo a dicho error. Una vez procesados todos los patrones de entrenamiento, comprobaremos la condición de parada del bucle externo y volveremos a empezar por el primer patrón, si la condición no se cumple.

- *Conjuntos de datos*: El algoritmo trabajará con un fichero de *entrenamiento* y un fichero de *test*. El ajuste de pesos se realizará utilizando los datos de entrenamiento y, en cada iteración del bucle externo, mostraremos el error cometido por la red en el conjunto de entrenamiento. La mejor forma de saber si el algoritmo ha sido correctamente implementado, es comprobar que este error de entrenamiento converge y tiende a hacerse cada vez más pequeño. Además, al terminar la ejecución del algoritmo, mostraremos el error cometido por la red en el fichero de *test*. En cualquier caso, los datos de *test* nunca deberán ser utilizados ni para ajustar los pesos, ni para decidir cuando detener el algoritmo.

- *Condición de parada*: El entrenamiento se detendrá si se produce alguna de estas tres condiciones:

  - Se han realizado más de $1000 \cdot N$ ajustes completos de pesos en la red, donde N es el número de patrones del conjunto de datos. Es decir, 1000 iteraciones para el bucle externo, cada una de las cuales supone N iteraciones del bucle interno (una iteración por patrón).
  - Si durante 50 iteraciones seguidas el error de entrenamiento no ha disminuido (o ha aumentado). Se debe utilizar una tolerancia de 10^{-5} para realizar esta comprobación, es decir, si el error de entrenamiento disminuye en una cantidad menor o igual que 10^{-5}, entonces consideramos que dicho error no ha disminuido.

- *Copias de los pesos*: Dependiendo del problema, la superficie de error puede ser muy compleja y a veces el algoritmo puede saltar a un punto donde no es capaz de moverse para minimizar el error. Es por ello que debemos mantener una "*copia de seguridad*" de los pesos de la red que han llevado, hasta el momento, al menor error. De manera que, antes de detener el algoritmo, siempre restauraremos esta copia de seguridad.

- *Semillas para los números aleatorios*: El algoritmo que estamos ejecutando es un algoritmo estocástico, es decir, la calidad de la red neuronal obtenida depende en gran medida del valor inicial de los pesos (primer punto explorado en la superficie de error). Para analizar mejor su comportamiento, vamos a intentar que el resultado no esté sesgado por la semilla de los números aleatorios. De otro modo, las conclusiones obtenidas pueden no ser válidas de forma general. Una forma de conseguirlo es efectuar varias ejecuciones con distintas semillas iniciales y calcular el resultado medio sobre todas las ejecuciones, para así representar con mayor fidelidad su comportamiento. Es por ello que el proceso de entrenamiento se repetirá cinco veces, utilizando las semillas 1, 2, 3, 4 y 5, y después de ello mostraremos la media y la desviación típica del error de entrenamiento y el error de *test* durante estas cinco ejecuciones.

3. Experimentos

Probaremos distintas configuraciones de la red neuronal y ejecutaremos cada configuración con cinco semillas (1, 2, 3, 4 y 5). A partir de los resultados obtenidos, se obtendrá la media y la desviación típica del error. Se deberá calcular el error MSE para el conjunto de entrenamiento y el error MSE para el conjunto de $test$. El MSE se define de la siguiente forma:

$$MSE = \frac{1}{N}\sum_{p=1}^{N}\left(\frac{1}{k}\sum_{o=1}^{k}(d_{po} - o_{po})^2\right), \tag{1}$$

donde N es el número de patrones de la base de datos considerada (entrenamiento o $test$), k es el número de salidas, d_{po} es el valor deseado para el patrón p y la variable de salida o y o_{po} es el valor obtenido.

Para valorar cómo funciona el algoritmo implementado en esta práctica, emplearemos un total de cuatro bases de datos:

- *Problema XOR*: esta base de datos representa el problema de clasificación no lineal del XOR. Se utilizará el mismo fichero para *train* y para $test$.

- *Función seno*: esta base de datos está compuesta por 120 patrones de *train* y 41 patrones de *test*. Ha sido obtenida añadiendo cierto ruido aleatorio a la función seno (ver Figura 1).

- *Base de datos quake*: esta base de datos está compuesta por 1633 patrones de *train* y 546 patrones de *test*. Se corresponde con una base de datos en la que el objetivo es averiguar la fuerza de un terremoto (medida en escala sismológica de Richter). Como variables de entrada, utilizamos la profundidad focal, la latitud en la que se produce y la longitud [1].

- *Base de datos Auto MPG*: El conjunto de datos Auto MPG, disponible en el Repositorio de Aprendizaje Automático de la UCI[2], es un conocido conjunto de datos utilizado frecuentemente en tareas de aprendizaje automático y análisis de datos. Contiene información sobre varios modelos de coches, incluidos sus atributos y la eficiencia del combustible. Este conjunto de datos se utiliza a menudo para tareas de regresión y modelado predictivo, en particular para predecir las millas por galón (MPG) de un coche en función de sus características (cilindros, cilindrada, potencia, etc.). El conjunto de datos original tiene 398 muestras y 7 atributos. Hemos binarizado los atributos discretos y eliminado el nombre del modelo, de modo que la lista final de atributos del conjunto de datos es: cilindros, cilindrada, potencia, peso, aceleración, año del modelo, EE.UU., Europa, Japón y MPG (variable objetivo).

Para aprovechar mejor las características de la función sigmoide, las variables de entrada han sido normalizadas en el intervalo $[-1, 1]$. La variable a predecir, ha sido escalada en el intervalo

[1] Para más información, consultar `https://sci2s.ugr.es/keel/dataset.php?cod=75`

[2] `https://archive.ics.uci.edu/dataset/9/auto+mpg`

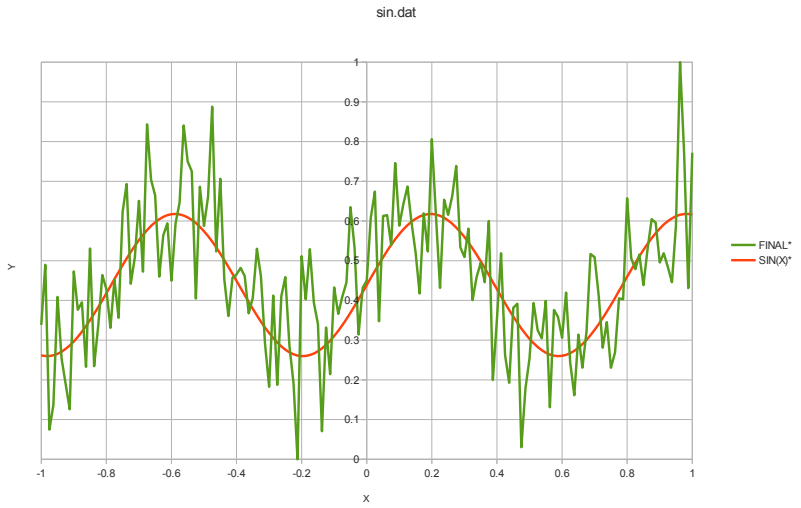

Figura 1: Representación de los datos incluidos para el problema de estimación de la función seno.

$[0, 1]$. **Ten en cuenta que en la base de datos Auto MPG deberá ser el código desarrollado el que realice la normalización de datos.**

Se deberá construir **una tabla para cada base de datos**, en la que se compare la media y la desviación típica del MSE de entrenamiento y de *test* (para el XOR, bastará de entrenamiento) para las distintas configuraciones utilizadas. Se deben probar, al menos, las siguientes configuraciones:

- *Arquitectura de la red*: Para esta primera parte, utilizaremos factor de momento. Se deberá probar un total de 12 arquitecturas:
  - Con una capa oculta: $\{n : 2 : k\}$, $\{n : 4 : k\}$, $\{n : 8 : k\}$, $\{n : 32 : k\}$, $\{n : 64 : k\}$ y $\{n : 100 : k\}$.
  - Con dos capas ocultas: $\{n : 2 : 2 : k\}$, $\{n : 4 : 4 : k\}$, $\{n : 8 : 8 : k\}$, $\{n : 32 : 32 : k\}$, $\{n : 64 : 64 : k\}$ y $\{n : 100 : 100 : k\}$.
- *Iteraciones*: Una vez decidida la mejor arquitectura para cada problema, probaremos con distinto número de iteraciones máximas: $i \in \{100; 500; 1000\}$.

Como valor orientativo, se muestra a continuación el error de entrenamiento y de generalización obtenido por una regresión lineal, utilizando Weka, en las cuatro bases de datos:

- *Problema XOR*: $MSE_{\text{train}} = MSE_{\text{test}} = 0{,}25$.
- *Función seno*: $MSE_{\text{train}} = 0{,}02968729$; $MSE_{\text{test}} = 0{,}03636649$.
- *Base de datos Quake*: $MSE_{\text{train}} = 0{,}03020644$; $MSE_{\text{test}} = 0{,}02732409$.
- *Base de datos Auto MPG*: $MSE_{\text{train}} = 0{,}00000$; $MSE_{\text{test}} = 0{,}00750$.

El alumno debería ser capaz de mejorar estos errores con algunas de las configuraciones.

3.1. Formato de los ficheros

Los ficheros que contienen las bases de datos tendrán el siguiente formato:

- En la primer línea se incluirá el número total de entradas del problema (n), el número total de salidas del problema (k) y el número total de patrones del fichero N.
- Luego tendremos una línea por patrón y cada línea tendrá $n + k$ valores reales. Para el patrón/línea p:
  - Los primeros n valores serán las entradas del patrón, es decir, $\mathbf{x}_p = \{x_{p1}, \ldots, x_{pn}\}$.
  - Los siguientes k valores serán las salidas deseadas del patrón, es decir, $\mathbf{d}_p = \{d_{p1}, \ldots, d_{pk}\}$.

Un ejemplo de este tipo de ficheros (en concreto, el del problema XOR) es el siguiente:

```
2 1 4
 1 -1 1
-1 -1 0
-1  1 1
 1  1 0
```

4. Entregables

Los ficheros a entregar serán los siguientes:

- Memoria de la práctica en un fichero `pdf` que describa el programa generado, incluya las tablas de resultados y analice estos resultados.
- Fichero ejecutable de la práctica y código fuente.

4.1. Memoria de la práctica

La memoria de la práctica deberá incluir, al menos, el siguiente contenido:

- Portada con el número de práctica, título de la práctica, asignatura, titulación, escuela, universidad, curso académico, nombre, DNI y correo electrónico del alumno.
- Índice del contenido de la memoria con numeración de las páginas.
- Descripción de los modelos de redes neuronales utilizados (arquitectura y organización en capas) (**máximo 1 carilla**).
- Descripción en pseudocódigo de los pasos del algoritmo de retropropagación y de todas aquellas operaciones relevantes. El pseudocódigo deberá forzosamente reflejar la implementación/el desarrollo realizados y no ser una descripción genérica extraída de las diapositivas de clase o de cualquier otra fuente (**máximo 3 carillas**).
- Experimentos y análisis de resultados:
  - Breve descripción de las bases de datos utilizadas.
  - Breve descripción de los valores de los parámetros considerados.
  - Resultados obtenidos, según el formato especificado en la sección anterior.
  - Análisis de resultados. El análisis deberá estar orientado a justificar los resultados obtenidos, en lugar de realizar un análisis meramente descriptivo de las tablas. Tener en cuenta que esta parte es decisiva en la nota de la práctica. Se valorará la inclusión de los siguientes elementos de comparación:
    - Gráficas de convergencia: reflejan, en el eje x, el número de iteración del algoritmo y, en el eje y, el valor del error de entrenamiento y el valor del error de *test*.

    - Análisis del modelo de red neuronal obtenido para el problema del XOR, utilizando la arquitectura más simple. Incluir el grafo del modelo, junto con el valor de todos los pesos. Comprobar cuál es el valor de las salidas obtenidas por este modelo frente al valor deseado.
    - Cualquier otro gráfico o análisis que el alumno estime oportuno (por ejemplo, se puede representar la aproximación de la función de seno realizada por el mejor modelo de red, utilizando un formato similar al de la Figura 1).

- Referencias bibliográficas u otro tipo de material distinto del proporcionado en la asignatura que se haya consultado para realizar la práctica (en caso de haberlo hecho).

Aunque lo importante es el contenido, se valorará también la presentación, incluyendo formato, estilo y estructuración del documento. La presencia de demasiadas faltas ortográficas puede disminuir la nota obtenida.

4.2. Ejecutable y código fuente

Junto con la memoria, se deberá incluir el fichero ejecutable preparado para funcionar en las máquinas de la UCO (en concreto, probar por `ssh` en `ts.uco.es`). Además se incluirá todo el código fuente necesario. El fichero ejecutable deberá tener las siguientes características:

- Su nombre será `la1`.
- El programa a desarrollar recibe ocho argumentos por la línea de comandos (que pueden aparecer en cualquier orden)[3]:
    - Argumento `t`: Indica el nombre del fichero que contiene los datos de entrenamiento a utilizar. Sin este argumento, el programa no puede funcionar.
    - Argumento `T`: Indica el nombre del fichero que contiene los datos de $test$ a utilizar. Si no se especifica este argumento, utilizar los datos de entrenamiento como $test$.
    - Argumento `i`: Indica el número de iteraciones del bucle externo a realizar. Si no se especifica, utilizar 1000 iteraciones.
    - Argumento `l`: Indica el número de capas ocultas del modelo de red neuronal. Si no se especifica, utilizar 1 capa oculta.
    - Argumento `h`: Indica el número de neuronas a introducir en cada una de las capas ocultas. Si no se especifica, utilizar 5 neuronas.
    - Argumento `e`: Indica el valor del parámetro eta (η). Por defecto, utilizar $\eta = 0{,}1$.
    - Argumento `m`: Indica el valor del parámetro mu (μ). Por defecto, utilizar $\mu = 0{,}9$.
    - Argumento `s`: Flag que indica que los datos (entrenamiento y test) van a normalizarse tras su lectura. Se normalizará en el intervalo $[-1, 1]$ las entradas y $[0, 1]$ la(s) salida(s).
- Opcionalmente, se puede añadir otro argumento para guardar la configuración del modelo entrenado (será necesario para hacer predicciones para la competición de Kaggle):
    - Argumento `w`: Indica el nombre del fichero en el que se almacenarán la configuración y el valor de los pesos del modelo entrenado.
- Un ejemplo de ejecución se puede ver en la siguiente salida:

```
1 i02gupep@NEWTS:~/imc/imc2021/workspace/la1/Debug$ ./la1 -t ../train_xor.dat -T ../
      test_xor.dat -i 1000 -l 1 -h 10 -e 0.1 -m 0.9
2 **********
3 SEED 1
4 **********
```

[3]Para procesar la secuencia de entrada, se recomienda utilizar la función `getopt()` de la librería `libc`

```
Iteration 1        Training error: 0.298534
Iteration 2        Training error: 0.287343
...
Iteration 1000     Training error: 0.00964613
===============
Layer 1
------
0.633863 -0.403033 0.773307
0.469708 0.669090 -0.478398
-0.344827 0.587652 -0.381660
-0.838194 0.833304 0.683623
-0.652504 0.552441 0.931250
1.037431 0.353168 0.532612
-0.936836 -0.204622 -0.874316
-2.507921 -2.810956 2.605680
-2.582238 -2.257553 -2.377693
-2.176769 2.170193 -2.377776
Layer 2
------
-0.291562 -0.309591 -0.436306 -1.468180 -0.887955 0.491371 -0.703266 4.389247
    -4.066886 2.783916 -0.831752
Desired output Vs Obtained output (test)
=========================================
1 -- 0.891937
0 -- 0.09749
1 -- 0.914304
0 -- 0.100294
We end!! => Final test error: 0.00964613
**********
SEED 2
**********
Iteration 1        Training error: 0.25546
Iteration 2        Training error: 0.254574
...
We end!! => Final test error: 0.00827856
**********
SEED 5
**********
Iteration 1        Training error: 0.268381
Iteration 2        Training error: 0.264984
...
Iteration 1000     Training error: 0.00813402
NETWORK WEIGHTS
===============
Layer 1
------
-2.621413 -2.489292 2.562251
-1.247278 -1.357582 -1.320136
-0.106533 -0.523225 -0.208347
1.382290 -1.410529 -1.396242
1.668992 -1.646763 -1.711076
-1.818481 -1.982124 -1.906150
-0.163134 -0.476374 -0.384511
0.097310 -0.311289 -0.365728
-1.168762 1.072397 1.266540
1.949929 -1.952045 1.882673
Layer 2
------
4.172039 -1.404449 -0.930334 1.370385 1.978051 -2.735954 -0.117170 -1.000121
    -0.972900 -2.964386 0.838221
Desired output Vs Obtained output (test)
=========================================
1 -- 0.915715
0 -- 0.0884255
1 -- 0.902992
0 -- 0.0905678
We end!! => Final test error: 0.00813402
```

```
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
************
Train error (Mean +- SD): 0.00895706 +- 0.000701482
Test error (Mean +- SD):          0.00895706 +- 0.000701482

i02gupep@NEWTS:~/imc/imc2021/workspace/la1/Debug$ ./la1 -t ../train_mpg.dat -T ../
    test_mpg.dat -i 1000 -l 2 -h 32 -e 0.1 -m 0.9 -s
P0:
6,225,110,3620,18.7,78,1,0,0,18.6,
P1:
4,140,92,2572,14.9,76,1,0,0,25,
P2:
6,171,97,2984,14.5,75,1,0,0,18,
P0:
0.2,-0.194805,-0.304348,0.138078,0.27381,0.333333,1,-1,-1,0.255319,
P1:
-0.6,-0.636364,-0.5,-0.456195,-0.178571,0,1,-1,-1,0.425532,
P2:
0.2,-0.475325,-0.445652,-0.222569,-0.22619,-0.166667,1,-1,-1,0.239362,
**********
SEED 1
**********
Iteration 1        Training error: 0.0135834
Iteration 2        Training error: 0.00987455
Iteration 3        Training error: 0.00858387
...
We end!! => Final test error: 0.00450654
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
************
Train error (Mean +- SD): 0.00369789 +- 0.000121391
Test error (Mean +- SD):          0.00444408 +- 0.000110289

i02gupep@NEWTS:~/imc/imc2021/workspace/la1/Debug$ ./la1 -t ../train_quake.dat -T
    ../test_quake.dat -i 1000 -l 1 -h 8 -e 0.1 -m 0.9
**********
SEED 1
**********
Iteration 1        Training error: 0.0303505
Iteration 2        Training error: 0.0303038
...
We end!! => Final test error: 0.0270451
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
************
Train error (Mean +- SD): 0.0297152 +- 6.14546e-05
Test error (Mean +- SD):          0.0269837 +- 6.04512e-05
```

4.3. [OPCIONAL] Obtención de predicciones para Kaggle

El mismo ejecutable de la práctica permitirá obtener las predicciones de salidas para un determinado conjunto de datos. Esta salida debe guardarse en un archivo `.csv` que deberéis subir a Kaggle para participar en la competición (ver el archivo `sampleSubmission.csv` en la plataforma Kaggle). Este modo de predicción, utiliza unos parámetros diferentes a los citados anteriormente:

- Argumento `p`: Flag que indica que el programa se ejecutará en modo de predicción.
- Argumento `T`: Indica el nombre del fichero que contiene los datos de *test* que se utilizará (`test_kaggle.dat`).
- Argumento `w`: Indica el nombre del fichero que contiene la configuración y los pesos del modelo entrenado que se utilizará para predecir las salidas.

A continuación, se muestra un ejemplo de ejecución del modo de entrenamiento haciendo uso del parámetro `w` para guardar la configuración del modelo.

```
i02gupep@NEWTS:~/imc/practica1/Debug$ ./la1 -t train.dat -T train.dat -w weights.txt -i
    100 -l 1 -h 32 -e 0.1 -m 0.9
**********
SEED 1
**********
Iteration 1         Training error: 0.015044
Iteration 2         Training error: 0.0135553
Iteration 3         Training error: 0.0118763
...
We end!! => Final test error: 0.00119128
WE HAVE FINISHED WITH ALL THE SEEDS
FINAL REPORT
************
Train error (Mean +- SD): 0.00140273 +- 0.000219986
Test error (Mean +- SD):          0.00144491 +- 0.000222772
```

A continuación, se muestra un ejemplo de salida del modo de predicción:

```
i02gupep@NEWTS:~/imc/practica1/Debug$ ./la1 -p -T test_kaggle.dat -w weights.txt
Id,Predicted
0,0.0220356
1,0.0820215
2,0.0241959
...
3497,0.103475
3498,0.00675393
3499,0.00506246
```