Aprendizaje Automático
Práctica 1

García Fernández, Antonio 100346053
García García, Alba María 100346091

Leganés G83

18 Marzo 2018

Índice

1. Introducción

En esta práctica se realizarán dos tareas. **La primera consiste en determinar qué acción debe realizar Mario con el objetivo de sobrevivir y pasarse el nivel**. Esta acción se seleccionará mediante un modelo de clasificación que será expuesto en la primera parte de este documento.

La segunda tarea consiste en **predecir, utilizando un modelo de regresión, el valor de la recompensa intermedia de Mario en *n+6*, *n+12* y *n+24***. Finalmente el agente final solo implantará el modelo para *n+24*.

2. Parte 1: Clasificación

En primer lugar, se propone generar un **modelo para determinar la siguiente acción (considerando avanzar a la derecha y saltar) a ejecutar por Mario** con el fin de sobrevivir y llegar al final del nivel, de modo que se pueda pasar el mayor número de niveles posible. Para lograr este objetivo, haremos uso de algoritmos de clasificación, que, tras observar el estado de el mundo de Mario concluirá a qué clase pertenece dicha instancia, es decir, que acción ha de realizar Mario.

En esta parte se explicará cómo se han tratado los datos que hemos proporcionado al agente así como la obtención de modelos a partir de dichos datos. Finalmente, se implementará el mejor modelo y se evaluarán sus resultados.

2.1. Balanceado de instancias

El primer problema que encontramos al intentar analizar las instancias es que en este dominio **las instancias recogidas están increíblemente desbalanceadas**. Esto se debe a que la mayoría de instancias son casos en los que Mario simplemente avanza hacia la derecha. Por supuesto, es importante que nuestro agente se desplace, pero también es crucial para superar un nivel que salte en los momentos adecuados.

Para superar esta traba impuesta por la naturaleza del problema aplicaremos un **filtro en el propio agente** que modifica qué instancias que son escritas en el fichero de entrenamiento. Las situaciones que hemos considerado interesantes para grabar y que, por tanto, estarán balanceadas entre ellas, habiendo un 20 % aproximadamente de cada una:

- Mario se encuentra en el suelo y **salta para evitar un enemigo** hacia el que se aproxima.
- Mario, estando en el suelo, se encuentra un **obstáculo** que le impide avanzar y **salta para superarlo**.
- Mario está **saltando un obstáculo** y una vez en el aire este sigue bloqueando su paso. Por tanto, **debe saltar más alto** (seguir pulsando *jump*) **para superarlo**.
- Mario **avanza hacia la derecha** estando en el suelo porque **no se encuentra con ningún enemigo**.
- Mario **avanza hacia la derecha** estando en el suelo porque **no se encuentra está bloqueado**.

Los dos últimos casos son mayoritarios. Sin embargo, los otros tres tampoco están balanceados entre ellos. La solución que terminamos adoptando es **únicamente escribir** en el archivo de entrenamiento estos casos y hacerlo siempre **asegurando que existe el mismo número de casos de cada tipo**.

No hemos considerado las instancias en las que Mario tiene un enemigo delante y está en el aire, ya que **queremos aquellos momentos en los que Mario salta tras ver a un enemigo**. Del mismo modo, **no nos interesan las acciones en las que Mario corre y no está en el suelo**, ya que corresponderán a momentos en los que cae del salto.

2.2. Descripción de los atributos seleccionados

Para generar un modelo de clasificación no son necesarios todos los datos que proporcionábamos desde el agente en el *Tutorial 3*. En esta sección describimos **qué atributos consideramos relevantes para la tarea de clasificación así como aquellos que hayan podido ser añadidos por nosotros** y las decisiones que nos han llevado a ello.

2.2.1. Evaluación con todos los atributos

Primero hemos querido comprobar **qué tal se comportan los algoritmos de clasificación con los atributos que se recogieron el *Tutorial 3***; quitando aquellos que refieren a la acción, ya que ahora la clase es la acción y no la distancia recorrida en el eje X; y eliminando también los atributos sobre el futuro, ya que estos sólo serán útiles para las predicciones que haremos en la segunda parte: regresión. Este sería el **punto de partida** a partir del cual hay que ir mejorando el comportamiento de Mario.

Los **algoritmos** que hemos considerado son **ID3, J48, PART y ZeroR**, ya que son los que hemos trabajado en prácticas anteriores para modelos de clasificación, excluyendo Ibk porque hace uso de *instance-based learning*, por lo que no construye unas reglas generalizadas tras analizar todas las instancias, sino que compara las instancias nuevas que van llegando con las que ya había para cada tick. ZeroR nos sirve para conocer el mínimo índice de aciertos que debe superar cualquier modelo, es decir, el mínimo admisible.

Hemos considerado **tres ficheros** de datos:

- ***entrenamiento_original_1***: contiene las instancias recogidas tras 1000 iteraciones con diferentes semillas para *P1BotAgent_original_1*, ejecutado con el script *P1Test_original_1* manualmente. Tras esto, fue discretizado con 10 *bins* en Weka para que ID3 pudiera trabajar con este conjunto de datos. El fichero con los datos discretizados se encuentra en *entrenamiento_original_1_disc*.
- ***entrenamiento_original_2***: contiene las instancias recogidas tras 100 iteraciones con las 100 primeras semillas del juego (0−99) ejecutando *P1HumanAgent_original_1*.Tras esto, fue discretizado con 10 *bins* en Weka para que ID3 pudiera trabajar con este conjunto de datos. El fichero con los datos discretizados se encuentra en *entrenamiento_original_2_disc*
- ***entrenamiento_original_3***: contiene un 75 % de los datos de *entrenamiento_original_1* y un 25 % de los de *entrenamiento_original_2*; ya que el bot necesita menos tiempo de reacción y no duda al hacer ninguna acción. Tras esto, fue discretizado con 10 *bins* en Weka para que ID3 pudiera trabajar con este conjunto de datos. El fichero con los datos discretizados se encuentra en *entrenamiento_original_3_disc*

Resultados para el bot

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 1000 instancias en cada hoja, ya que tenemos en total **20.217 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	ID3	J48	PART	ZeroR
entrenamiento_original_1_disc.arff	67,70	78,12 v	77,17 v	62,23 *
	(v/ /*)	(0/1/0)	(1/0/0)	(0/0/1)

Cuadro 1: Resultados del análisis del experimento para *entrenamiento_original_1_disc.arff*

Podemos ver que **ZeroR da un valor de aciertos exageradamente alto**. Esto se debe a que **el bot siempre se mueve a la derecha**, por lo que las únicas clases que aparecen en las instancias son *move* y *move_jump*, correspondiendo los saltos a 3/5 de las situaciones consideradas al balancear, es decir, ese 62,23 %. Cuando introduzcamos instancias del entrenamiento del humano, encontraremos las cuatro clases en las instancias.

El resto de los algoritmos mejora notablemente con respecto a ZeroR, **sobresaliendo J48 y PART** entre ellos. Iremos viendo en los siguientes análisis si sigue siendo la mejor opción.

Respecto a las clases a las que pertenecen las instancias, tenemos los siguientes datos:

- La acción más concurrida es ***move_jump***, con 12.582 instancias. (62, 23 %)
- La siguiente acción más usada es ***move***, con 7.635 instancias. (37,77 %)

Como podemos observar, se mantiene la relación a 3/5 de las instancias de aquellas que contienen salto con las que no.

Resultados para el humano

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 85 instancias en cada hoja, ya que teníamos en total **1.734 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	ID3	J48	PART	ZeroR
entrenamiento_original_2_disc.arff	47,77	60,97 v	58,49 v	48,90
	(v/ /*)	(1/0/0)	(1/0/0)	(0/1/0)

Cuadro 2: Resultados del análisis del experimento para *entrenamiento_original_2.arff*

Como era de esperar, **las instancias correctamente clasificadas descienden notablemente con los datos recogidos del humano**, ya que sus acciones dependen de muchos más factores que el bot y no siempre toma la misma decisión en la misma situación; a pesar de que se intentó jugar como lo hace el bot: sin lanzar bolas de juego, sin ir hacia atrás, etcétera.

Como punto a tener en cuenta, podemos ver que **ID3 no es muy bueno clasificando instancias en esta situación**, ya que es incluso peor que ZeroR, es decir, el mínimo número de aciertos admisible; que ha descendido del caso anterior, como era de suponer. Por el contrario, **J48 y PART destacan como las mejores opciones**.

Respecto a las clases a las que pertenecen las instancias, tenemos los siguientes datos:

- La acción más concurrida es ***move_jump***, con 848 instancias. (48, 90 %)
- La siguiente acción más usada es ***move***, con 365 instancias. (21, 05 %)
- Después nos encontramos con ***still***, que tiene 295 instancias. (17,01 %)
- Y, finalmente, ***jump***, con 226 instancias. (13,03 %)

Como podemos observar, se mantiene la relación a 3/5 de las instancias de aquellas que contienen salto con las que no. Y que, con excepción de *move_jump*, el resto de instancias están más o menos balanceadas.

Resultados para el fichero mixto bot-humano

Para la selección de instancias de este fichero, decidimos coger un **25 % de instancias del humano y un 75 % del bot**. En el caso del humano las cogimos todas, ya que no eran muchas (1.734); por lo que quedarían 5202 del bot.

Estas 6.936 instancias las hemos querido recoger de forma distribuida de entre todas las disponibles, de modo que cogiera diferentes semillas (muchas de ellas no aparecen representadas en el humano) y que así los datos fueran más representativos. De este modo, recogimos, empezando por la primera, bloques de 500 instancias separados por bloques de 1200 que son ignoradas. Una vez creado el fichero, las discretizamos conjuntamente.

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 350 instancias en cada hoja, ya que teníamos en total **6.936 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	**ID3**	**J48**	**PART**	**ZeroR**
entrenamiento_original_3_disc.arff	62,30	72,79 v	72,59 v	59,00 *
	(v/ /*)	(1/0/0)	(1/0/0)	(0/0/1)

Cuadro 3: Resultados del análisis del experimento para *entrenamiento_original_3.arff*

Como podemos ver, los resultados son bastante mejores que con datos sólo del humano y un poco peores que con el bot. Esto nos indica claramente, que **la mejor fuente de aprendizaje para Mario son los datos del bot**, aunque siempre es interesante añadir un porcentaje pequeño de los del humano en el modelo, ya que añade situaciones que en el bot no se dan (*jump* y *still*).

Por otro lado, seguimos viendo que se mantiene la **prevalencia de J48 y PART** sobre ID3 en cuanto a resultados.

2.2.2. Evaluación tras la selección de atributos

Usando el fichero *entrenamiento_original_3.arff*, el **árbol que obtenemos** tras realizar una validación cruzada de 10 *folds* con el algoritmo que mejores resultados ha dado hasta ahora, **J48**, aplicando previamente una pre-poda de 350 elementos por hoja como en el caso anterior; es el siguiente:

if (distancePassedCells $\leq$ 4) **move**
if (distancePassedCells $>$ 4)
| if (timeSpent $\leq$ 3) **move_jump**
| if (timeSpent $>$ 3)
| | if (MergedObs[10][8] = -85) **move_jump**
| | if (MergedObs[10][8] = -62) **move_jump**
| | if (MergedObs[10][8] = -60) **move_jump**
| | if (MergedObs[10][8] = -24) **move_jump**
| | if (MergedObs[10][8] = 0) **move**
| | if (MergedObs[10][8] = 1) **move**
| | if (MergedObs[10][8] = 2) **move_jump**
| | if (MergedObs[10][8] = 3) **move_jump**
| | if (MergedObs[10][8] = 5) **move_jump**
| | if (MergedObs[10][8] = 25) **move_jump**
| | if (MergedObs[10][8] = 80) **move_jump**
| | if (MergedObs[10][8] = 93) **move_jump**

Viendo los resultados podemos obtener dos conclusiones claras:

- **Sólo aparecen las clases *move_jump* y *move***, ya que las otras dos sólo las ejecuta el humano, que representa un 25 % de las instancias, por lo que han resultado ser minoritarias. Esto **no representa un problema**, ya que consigue que Mario esté siempre en movimiento; y teniendo en cuenta que **es peligroso que se pare**, ya que puede recibir la misma información del entorno si no se mueve y quedarse en un punto del mapa indefinidamente.
- Los **atributos más informados** de los que tenemos actualmente para que Mario tome una acción u otra viene del **entorno que le rodea (*MergedObs*)**, aunque sólo para las posiciones cercanas a él, como podemos ver en el árbol, que sólo utiliza la posición [10][8], siendo la posición de Mario [9][9].

Analizando los **atributos** que tiene en cuenta el árbol, podemos ver que hay dos que resultan un tanto **extraños**. Estos son *distancePassedCells* y *timeSpent*:

- ***distancePassedCells***. Se da la coincidencia que **en la mayoría de los mapas Mario no encuentra ningún enemigo ni obstáculo en las primeras cuatro celdas**, por lo que no necesita saltar y simplemente se mueve. Este dato no es representativo, ya que la configuración del mapa no tiene por qué ser así.
- ***timeSpent***. En los tres primeros segundos de juego parece que Mario salta mientras se mueve a la derecha de forma regular, pero **no encontramos ninguna relación entre este atributo y la clase**.

Tras eliminar los atributos *distancePassedCells* y *timeSpent*, obtenemos un **árbol análogo**, sólo que **en el lugar de *distancePassedCells* tenemos *distancePassedPhys***, cambiando 4 por 77, que suponemos que será el valor análogo y **en el lugar de *timeSpent*, tenemos *timeLeft***, invirtiendo la condición de los tres primeros segundos a la de cuando quedan más de 194. Debemos eliminar, por tanto, estos dos atributos también.

Tras su eliminación, vemos que se cuela en el árbol otro atributo relacionado con la posición. Esta vez se trata de ***MarioPosX***, que es una variable que representa lo mismo que *distancePassedPhys*, pero con números decimales en lugar de enteros. Por esta misma razón, **no es una variable que deberíamos tener en cuenta**.

Tras eliminar todas estas variables que suponían un peligro de que Mario pudiera memorizar los mapas, obtenemos un **árbol** que sólo tiene en cuenta **algunas celdas de *MergedObs***:

- Posición [10][11], una celda debajo de Mario y dos hacia la derecha.
- Posición [10][4], una celda debajo de Mario y cinco detrás.
- Posición [9][10], una celda delante de Mario.

El resultado de **instancias correctamente clasificadas en este modelo es del 72,79** %, por lo que eliminar los anteriores atributos no afecta en absoluto a la clasificación. Las reglas generadas son las siguientes:

if (MergedObs[10][11] = -85) **move_jump**
if (MergedObs[10][11] = -62) **move_jump**
if (MergedObs[10][11] = -60)
| if (MergedObs[10][4] = -85) **move_jump**
| if (MergedObs[10][4] = -62) **move**
| if (MergedObs[10][4] = -60) **move_jump**
| if (MergedObs[10][4] = -24) **move_jump**
| if (MergedObs[10][14] = 0)
| | if (MergedObs[9][10] = -85) **move_jump**
| | if (MergedObs[9][10] = -62) **move**
| | if (MergedObs[9][10] = -60) **move_jump**
| | if (MergedObs[9][10] = -24) **move_jump**
| | if (MergedObs[9][10] = 0) **move**
| | if (MergedObs[9][10] = 1) **move_jump**
| | if (MergedObs[9][10] = 2) **move**
| | if (MergedObs[9][10] = 3) **move**
| | if (MergedObs[9][10] = 5) **move**
| | if (MergedObs[9][10] = 25) **move**
| | if (MergedObs[9][10] = 80) **move**
| | if (MergedObs[9][10] = 93) **move**
| if (MergedObs[10][4] = 1) **move_jump**
| if (MergedObs[10][4] = 2) **move_jump**
| if (MergedObs[10][4] = 3) **move_jump**
| if (MergedObs[10][4] = 5) **move_jump**
| if (MergedObs[10][4] = 25) **move_jump**
| if (MergedObs[10][4] = 80) **move_jump**
| if (MergedObs[10][4] = 93) **move_jump**
if (MergedObs[10][11] = -24) **move**
if (MergedObs[10][11] = 0) **move**
if (MergedObs[10][11] = 1) **move**
if (MergedObs[10][11] = 2) **move_jump**
if (MergedObs[10][11] = 3) **move_jump**
if (MergedObs[10][11] = 5) **move_jump**
if (MergedObs[10][11] = 25) **move_jump**
if (MergedObs[10][11] = 80) **move_jump**

if (MergedObs[10][11] = 93) **move_jump**

Tras ver el resultado del árbol, eliminaremos todos los atributos que usaban *P1BotAgent_original* y *P1HumanAgent_original* para dejar sólo la **casilla más cercana a Mario en todas las direcciones** (incluyendo diagonales) **en *MergedObs*, a excepción de las tres de detrás**, ya que Mario nunca se moverá hacia atrás; **incluyendo además una casilla adicional para las direcciones *delante* y *arriba*** ya que son en las dos direcciones en las que el agente se moverá. Hemos considerado hasta dos celdas de la posición de Mario porque **nuestro bot es capaz de moverse con la información de la celda inmediata y la siguiente**.

Resultados para el bot

Probamos primero estos atributos en el bot, guardando los resultados en *entrenamiento_filtrado_1.arff*, un fichero que contiene las mismas instancias que *entrenamiento_original_1.arff*, pero dejando sólo los atributos que nos interesan. En este caso **no necesitamos discretizar**, ya que todos los atributos son nominales.

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 1000 instancias en cada hoja, ya que teníamos en total **20.507 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	**ID3**	**J48**	**PART**	**ZeroR**
entrenamiento_filtrado_1.arff	82,53	81,48 v	82,49 v	62,23 *
	(v/ /*)	(1/0/0)	(1/0/0)	(0/0/1)

Cuadro 4: Resultados del análisis del experimento para *entrenamiento_filtrado_1.arff*

Como podemos ver, los resultados mejoran notablemente en todos los algoritmos con respecto a los resultados del bot con todos los atributos del *Tutorial 3*:

- **ID3** pasa de un 67,70 % a un 82,53 % de aciertos.
- **J48** pasa de un 78,12 % a un 81,48 % de aciertos.
- **PART** pasa de un 77,17 % a un 82,49 % de aciertos.

Además de la mejora, todos están por encima del umbral marcado por **ZeroR** (62,23 %) e incluso encima de un 80 % de aciertos de forma general. Asimismo, vemos que **el desequilibro que existía entre ID3 con J48 y PART ha desaparecido prácticamente**, igualándose los tres (quizás debido a que ya no manejamos atributos numéricos).

De este modo, concluimos que ha sido un acierto eliminar todas aquellas instancias y que parece ser que estas que estamos usando **ayudan a Mario a aprender**.

Resultados para el humano

Probamos después estos atributos en el humano, guardando los resultados en *entrenamiento_filtrado_2 .arff*, un fichero que contiene las mismas instancias que *entrenamiento_original_2.arff*, pero dejando sólo los atributos que nos interesan. En este caso **tampoco necesitamos discretizar**, ya que todos los atributos son nominales.

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 85 instancias en cada hoja, ya que teníamos en total **1.734 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	**ID3**	**J48**	**PART**	**ZeroR**
entrenamiento_filtrado_2.arff	59,85	60,44	57,55 *	48,90 *
	(v/ /*)	(0/1/0)	(0/0/1)	(0/0/1)

Cuadro 5: Resultados del análisis del experimento para *entrenamiento_filtrado_2.arff*

- **ID3** pasa de un 47,77 % a un 59,85 % de aciertos.
- **J48** pasa de un 60,97 % a un 60,44 % de aciertos.
- **PART** pasa de un 58,49 % a un 57,55 % de aciertos.

Como podemos ver, **los resultados no han variado mucho, lo que significa que los atributos eliminados no contribuían de manera significativa a la tarea de clasificación**. Ha de destacarse la mejora de **ID3**, pero creemos que esto se debe a que a eliminar los atributos numéricos, ya que al aplicar cualquier tipo de filtro siempre se pierde precisión en la clasificación.

Creemos que esta **ausencia de una gran mejora** se debe a que es **mucho más difícil clasificar las acciones del humano** ya que están basadas en diferentes motivos que no siempre son las celdas que lo rodean (monedas, rutas alternativas por encima de los bloques, recoger un *power-up*, etc).

Resultados para el fichero mixto bot-humano

Tras esto, creamos un fichero compuesto de un **25 % de las instancias del humano y un 75 % de las del bot**, con las mismas instancias que *entrenamiento_original_3*, pero dejando sólo los atributos que indicamos al principio de la sección. Estas instancias se encuentran en *entrenamiento_filtrado_3 .arff*, un fichero de **6.936 instancias**. En este caso **tampoco necesitamos discretizar**, ya que todos los atributos son nominales.

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 350 instancias en cada hoja, ya que teníamos en total **6.936 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	ID3	J48	PART	ZeroR
entrenamiento_filtrado_3.arff	76,33	76,17 v	74,22 v	59,00 *
	(v/ /*)	(0/1/0)	(0/1/0)	(0/0/1)

Cuadro 6: Resultados del análisis del experimento para *entrenamiento_filtrado_3.arff*

- **ID3** pasa de un 62,30 % a un 76,33 % de aciertos.
- **J48** pasa de un 72,79 % a un 76,17 % de aciertos.
- **PART** pasa de un 72,59 % a un 74,22 % de aciertos.

Como se vio en los dos anteriores casos, **ID3 mejora notablemente** acercándose al nivel de acierto de **J48 y PART**, que **mejoran ligeramente**.

Vamos a analizar el **árbol que obtenemos usando J48** con las mismas condiciones para las que realizamos el experimento:

if (MergedObs[9][11] = -85) **move_jump**
if (MergedObs[9][11] = -62) **move**
if (MergedObs[9][11] = -60) **move_jump**
if (MergedObs[9][11] = -24) **move_jump**
if (MergedObs[9][11] = 0)
| if (MergedObs[9][10] = -85) **move**
| if (MergedObs[9][10] = -62) **move_jump**
| if (MergedObs[9][10] = -60) **move**
| if (MergedObs[9][10] = -24) **move_jump**
| if (MergedObs[9][10] = 0) **move**
| if (MergedObs[9][10] = 1) **move_jump**
| if (MergedObs[9][10] = 2) **move**
| if (MergedObs[9][10] = 3) **move**
| if (MergedObs[9][10] = 5) **move**
| if (MergedObs[9][10] = 25) **move**
| if (MergedObs[9][10] = 80) **move**
| if (MergedObs[9][10] = 93) **move**
if (MergedObs[9][11] = 1) **move_jump**
if (MergedObs[9][11] = 2) **move**
if (MergedObs[9][11] = 3) **move_jump**
if (MergedObs[9][11] = 5) **move**
if (MergedObs[9][11] = 25) **move_jump**
if (MergedObs[9][11] = 80) **move_jump**
if (MergedObs[9][11] = 93) **move_jump**

Se pueden sacar dos claras conclusiones:

- Al igual que en el árbol de decisión anterior, vemos que propicia que **Mario siempre se mueva**, ya que las clases *jump* y *still* no aparecen; debido a que sólo aparecen en el humano, cuyo peso en la cantidad de instancias es de sólo un 25 %.
- **Mario sólo se fija en la casilla de delante** ([9][10]) **y dos delante de él** ([9][11]) para saber si saltar o no; ya que se mueve siempre hacia delante. Esto **coincide con la forma que tiene el algoritmo del bot para moverse por los mapas**, lo que indica que vamos por buen camino.

A continuación, probaremos a incluir algún **atributo más informado** para mejorar el rendimiento del agente.

2.2.3. Evaluación tras añadir nuevos atributos más informados

Hemos considerado añadir los siguientes atributos:

- ***isBlocked***: este atributo vale *Blocked* si Mario está bloqueado por algún obstáculo que se encuentre en la casilla delante de él y *Free* en el caso contrario.
- ***isEnemy***: este atributo vale *Enemy* si Mario tiene algún enemigo una o dos casillas delante de él y *NoEnemy* en el caso contrario.
- ***distanceEnemy***: este atributo nos indica la distancia de Mario al enemigo más cercano, teniendo en cuenta toda la matriz de observación y utilizando la distancia euclídea.
- ***isOnGround***: este atributo nos indica si Mario se encuentra en el suelo (*Ground*) o en el aire (*Air*) para ese tick.
- ***jumped***: este atributo nos indica si Mario saltó en el tick pasado (*Jumped*) o no (*NoJumped*), lo que es muy útil para que Mario no se quede pulsando la tecla de salto una vez está en el suelo, ya que esto no permite que salte cuando sea necesario.

Como podemos ver, **hemos incluido todos aquellos atributos que hemos utilizado para crear el comportamiento de nuestro agente procedural** (*Tutorial 1*); para conseguir que aprenda con los mismos datos con los que programamos su comportamiento en un inicio, consiguiendo quizá, alguna relación nueva que no tuvimos en cuenta.

Resultados para el bot

Probamos primero estos atributos en el bot, guardando los resultados en *entrenamiento_new _attributes _1.arff*, un fichero de **20.555 instancias**. Este fichero lo obtuvimos tras ejecutar el *script P1Test_newAttributes* para la clase *P1BotAgent_newAttributes*. En este caso **sí que necesitamos discretizar para que se pueda ejecutar en ID3**, no como en los casos anteriores. Guardamos estos datos discretizados en *entrenamiento_new_attributes_1_disc.arff*

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 1000 instancias en cada hoja, ya que teníamos en total **20.555 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	ID3	J48	PART	ZeroR
entrenamiento_new_attributes_1_disc.arff	99,99	100,00 v	99,53 *	62,22 *
	(v/ /*)	(1/0/0)	(0/0/1)	(0/0/1)

Cuadro 7: Resultados del análisis del experimento para *entrenamiento_new_attributes_1_disc.arff*

Como podemos ver, los resultados mejoran notablemente en todos los algoritmos con respecto a los resultados del bot con el filtrado de atributos de los que había en el *Tutorial 3*:

- **ID3** pasa de un 82,53 % a un 99,99 % de aciertos.
- **J48** pasa de un 81,48 % a un 100,00 % de aciertos.
- **PART** pasa de un 82,49 % a un 99,53 % de aciertos.

A pesar de que tras incluir los nuevos atributos más informados **obtenemos casi un 100 % de aciertos**, esto se debe a que incluimos los atributos en los que se basa el bot para comportarse. Aun así, el hecho de que **haya sido capaz de reconocer los patrones de Mario a la hora de moverse**, se puede considerar todo un éxito. Sin embargo, en el caso de que el comportamiento fuera ligeramente diferente a la hora de jugar, ya no obtendríamos estos resultados, como veremos en el caso del humano.

Resultados para el humano

Tras el bot, toca el turno del humano; guardando los resultados en *entrenamiento_new _attributes _2.arff*, un fichero de **1.657 instancias**. Este fichero lo obtuvimos tras ejecutar manualmente *P1HumanAgent_newAttributes* en 100 ocasiones usando las 100 primeras semillas (0−99). Como en el caso anterior, **también necesitamos discretizar para que se pueda ejecutar en ID3**. Guardamos estos datos discretizados en *entrenamiento_new_attributes_2_disc.arff*

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 85 instancias en cada hoja, ya que teníamos en total **1.657 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	ID3	J48	PART	ZeroR
entrenamiento_new_attributes_2_disc.arff	69,34	74,17 v	74,17 v	57,82 *
	(v/ /*)	(1/0/0)	(1/0/0)	(0/0/1)

Cuadro 8: Resultados del análisis del experimento para *entrenamiento_new_attributes_2_disc.arff*

- **ID3** pasa de un 59,85 % a un 69,34 % de aciertos.
- **J48** pasa de un 60,44 % a un 74,17 % de aciertos.
- **PART** pasa de un 57,55 % a un 74,17 % de aciertos.

Con el uso de estos nuevos atributos hemos conseguido una **mejora general muy notable en los tres algoritmos que estamos evaluado**, a pesar de que se sigue quedando muy **lejos de los resultados del bot**, como ya previmos; debido a que el comportamiento del humano es más complejo e incluso contradictorio a veces.

Tras esto, vemos que **para el caso del humano las mejores opciones son J48 y PART**, quedándose ID3 un poco alejado de sus resultados, como viene sucediendo desde el principio; a pesar de que **para el bot eran igual de buenos los tres**.

Resultados para el fichero mixto bot-humano

Tras esto, creamos un **fichero compuesto de un 25 % de las instancias del humano y un 75 % de las del bot**, con una estructura similar a *entrenamiento_original_3*, cambiando los huecos de 1200 instancias por unos huecos de 1700 en el caso de las instancias del bot; ya que el número de instancias recogidas ha sido mayor en este caso. Estas instancias se encuentran en *entrenamiento_new_attributes_3.arff*, un fichero de **6.628 instancias**. Una vez creado dicho fichero, **lo discretizamos para poder probarlo luego en ID3**, guardándolo en *entrenamiento_new_attributes_3_disc.arff*.

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de J48 y PART elegimos hacer una pre-poda con un mínimo de 350 instancias en cada hoja, ya que teníamos en total **6.628 instancias**, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Dataset	ID3	J48	PART	ZeroR
entrenamiento_new_attributes_3.arff	92,08	93,60 v	93,60 v	61,35 *
	(v/ /*)	(0/1/0)	(0/1/0)	(0/0/1)

Cuadro 9: Resultados del análisis del experimento para *entrenamiento_new_attributes_3_disc.arff*

- **ID3** pasa de un 76,33 % a un 92,08 % de aciertos.
- **J48** pasa de un 76,17 % a un 93,60 % de aciertos.
- **PART** pasa de un 74,22 % a un 93,60 % de aciertos.

Mezclando datos de bot y de humano, obtenemos unos **muy buenos resultados**, como se ha estado viendo tras añadir los nuevos atributos.

Tras esto, hemos pensado **implementar tanto el modelo que obtengamos con las instancias del bot como el que tiene tanto instancias del bot como humanas**; ya que ambos dan muy buenos resultados.

2.3. Justificación del algoritmo seleccionado

Tras **varias pruebas para los algoritmos ID3, J48 y PART** podemos ver como, independientemente del conjunto de entrenamiento que estemos utilizando, J48 obtiene casi siempre los resultados más altos; aunque empatando con PART en dos ocasiones. Por este motivo, **decidimos utilizar el algoritmo J48 para la creación de nuestro modelo final**.

A continuación, se muestran unos histogramas que reflejan la **información obtenida en las anteriores secciones**. Son los datos en los que hemos basado nuestra decisión.

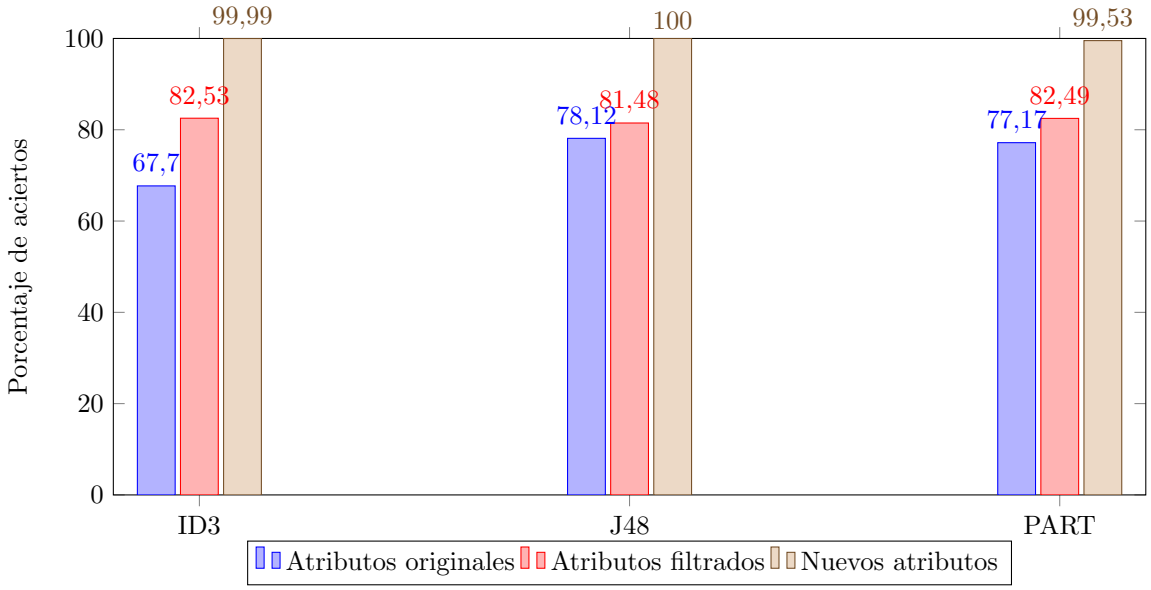

Figura 1: Variación del porcentaje de aciertos para **bot**, en los distintos algoritmos propuestos

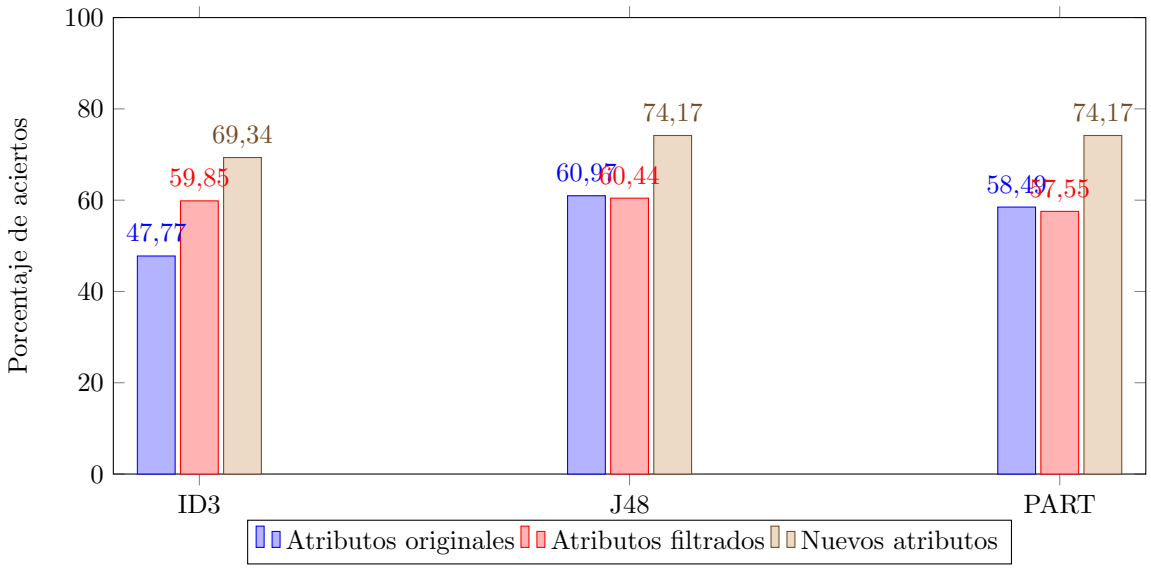

Figura 2: Variación del porcentaje de aciertos para el **humano**, en los distintos algoritmos propuestos

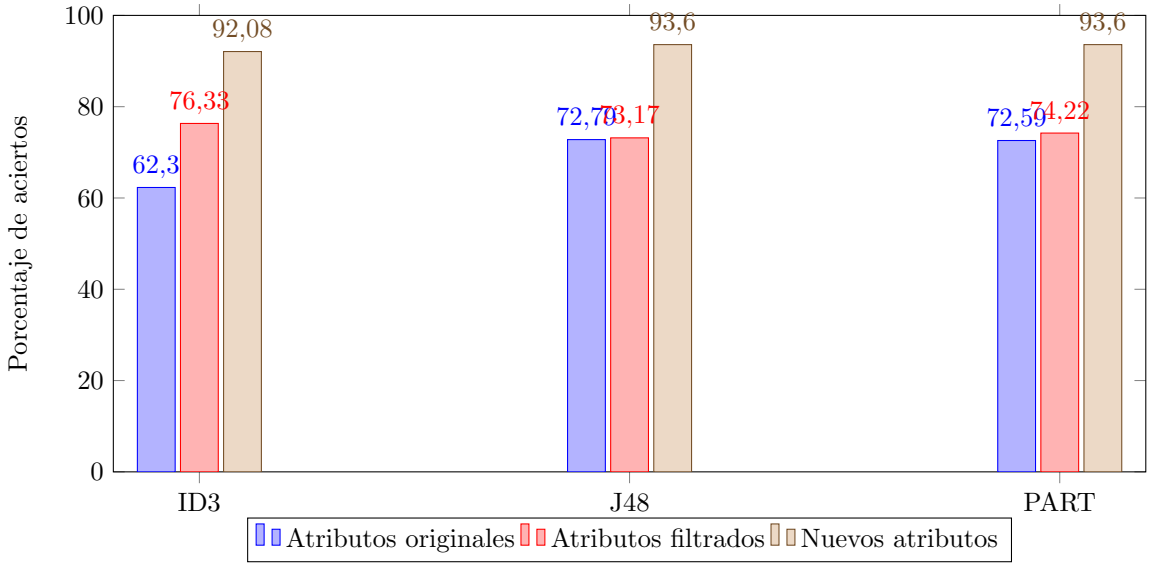

Figura 3: Variación del porcentaje de aciertos para el **mixto**, en los distintos algoritmos propuestos

2.4. Descripción de la implementación

Teniendo en cuenta los datos presentados en la sección anterior decidimos implementar **dos agentes**: usando tanto el **modelo generado por el bot** (*P1BotAgent_pureBot*) como **el generado por la mezcla de humano y bot** (*P1BotAgent_mixedBotHuman*), tal y como indicamos anteriormente.

Lo primero que planteamos en este problema fue el **balanceo de instancias** para las cinco situaciones básicas que se puede encontrar Mario mientras juega. Para implementarlo, **creamos un contador para cada una de las situaciones: *air_jump_counter*, *enemy_jump_counter*, *block_jump_counter*, *no_enemy_jump_counter* y *no_block_jump_counter***. En el tick 0 inicializamos *no_enemy_jump_counter* a 1 (cualquiera de los dos últimos valía, ya que son las instancias más abundantes) de modo que una vez que se encuentre con una del resto de las otras tres, el contador de *no_enemy_jump_counter* se actualizará y pasaremos a la situación anterior de nuevo. De este modo, **nos aseguramos de que grabamos el mismo número de instancias de cada una de las situaciones anteriores**.

Recoger los nuevos atributos fue una tarea sencilla. Afortunadamente, todos los atributos que añadimos excepto la distancia a un enemigo ya los habíamos creado inconscientemente para el *Tutorial 1*. Siguiendo el consejo del profesor, **identificamos estos atributos y simplemente tuvimos que añadirlos a la línea que se escribe en el fichero de entrenamiento**.

***isEnemy*, *isBlocked* e *isOnGround* eran booleanos que nuestro *getAction()* utilizaba para determinar la acción a llevar a cabo por Mario**. Dichas variables son actualizadas en el método *checkConditions()* haciendo uso de la matriz *mergedObservation[][]*, por lo que en este método añadimos el valor que ha tomado dicho booleano a la línea correspondiente de ese tick en el fichero de entrenamiento.

***jumped* es un booleano que indica si Mario ha saltado en el tick anterior**. En los agentes bot entregados en los tutoriales su función era prevenir que se apretara constantemente el botón de salto. Para que mantuviera el valor del tick anterior, siendo una variable global, únicamente lo escribimos en la línea correspondiente al tick actual antes de actualizar su valor para dicho tick.

distanceEnemy es el único que tuvimos que implementar desde cero. Para calcularlo, primero traversamos la región que se encuentra en frente de Mario en la matriz *mergedObservation[][]*, registrando el **enemigo más cercano tomando su distancia a Mario tanto en el eje horizontal como en el vertical**. **De estas dos distancias se obtiene mediante *getEuclideanDistance()* la distancia euclídea entre Mario y el enemigo**. Hay que puntualizar que escogimos la distancia euclídea frente a la de Manhattan debido a que en este problema los enemigos se pueden mover en diagonal: por ejemplo, al aer desde una plataforma superior mientras siguen desplázandose a la izquierda.

Los **árboles implementados** son de los siguientes:

Modelo generado por instancias de bot

```
if (isOnGround = Ground)
|  if (jumped = Jumped) move
|  if (jumped = NoJumped)
|  |  if (isBlocked = Blocked) move_jump
|  |  if (isBlocked = Free)
|  |  |  if (isEnemy = Enemy) move_jump
|  |  |  if (isEnemy = NoEnemy) move
if (isOnGround = Air) move_jump
```

Modelo generado por instancias mixtas

```
if (isOnGround = Ground)
|  if (isEnemy = Enemy) move_jump
|  if (isEnemy = NoEnemy)
|  |  if (isBlocked = Blocked)
|  |  |  if (jumped = Jumped) move
|  |  |  if (jumped = NoJumped) move_jump
|  |  if (isBlocked = Free) move
if (isOnGround = Air) move_jump
```

Como podemos observar, **Mario no hace uso de ninguna de las variables de *MergedObs***, debido a que las nuevas que introducimos están mucho más informadas (haciendo uso de un aprendizaje atributo-valor no se puede saber la relación entre las celdas de dicha matriz). Además, a pesar de que creíamos que *distanceEnemy* podría más informado que *isEnemy*, ha resultado que **Mario prefiere saber si tiene un enemigo delante que la distancia real a los enemigos**, quizá se debe a que *isEnemy* indica implítamente la dirección del enemigo (delante) y *distanceEnemy* no.

2.5. Evaluación de los resultados

A continuación se muestra una **gráfica indicando la distancia avanzada por los agentes *BaselineAgent*, *T1BotAgent*, *P1BotAgent_pureBot* y *P1BotAgent_mixedBotHuman*** en semillas que no forman parte del conjunto que se utilizó para el entrenamiento ([0..1000]). Las semillas que utilizamos fueron desde la 1001 a la 1026.

Como se puede apreciar en la gráfica, ***P1BotAgent_pureBot* obtiene los mejores resultados si se mide la distancia avanzada**; ya que llega a ganar 7 partidas de las 26 probadas (26,92 %). Su más próximo competidor es *T1BotAgent* con 2 partidas ganadas (7,69 %). En cuanto al resto de agentes, ninguno llega a ganar ninguna partida, lo que nos indica que ***P1BotAgent_mixedBotHuman* es peor de lo que pensábamos**

Tras esta evaluación **decidimos presentar el agente *P1BotAgent_pureBot*** que implementa el árbol de decisión de creado mediante el **algoritmo J48** con **ejemplos únicamente extraídos del bot** implemetado en el *Tutorial 1* (*entrenamiento_new_atributtes_1.arff*).

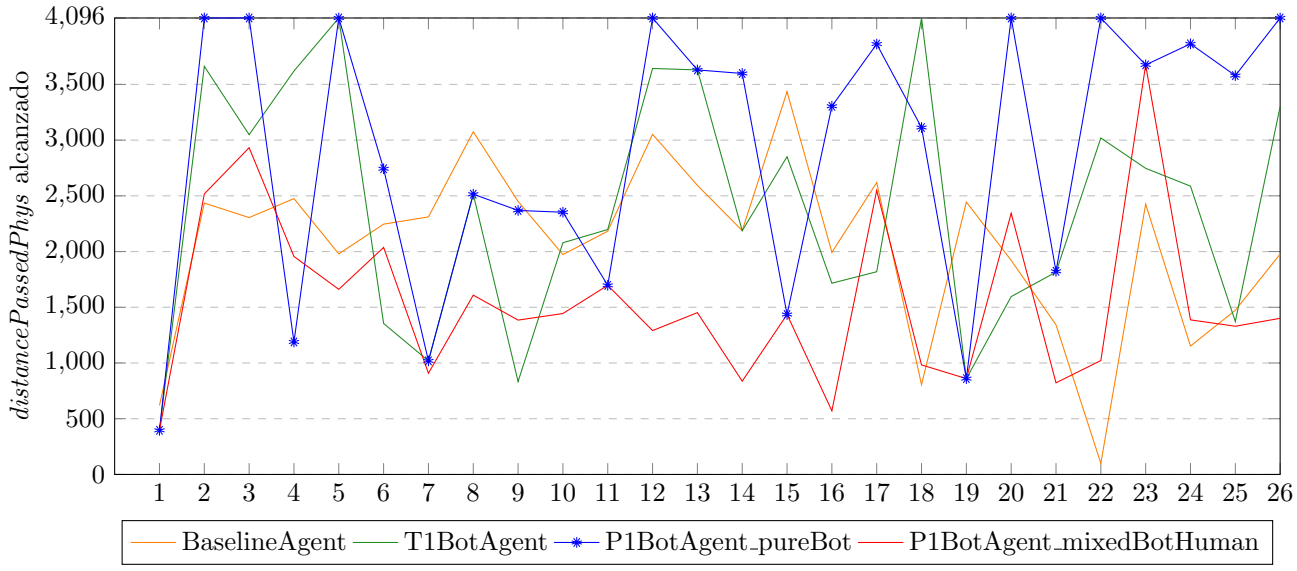

Figura 4: Comparación y evaluación de agentes en semillas nunca antes entrenadas

3. Parte 2: Regresión

3.1. Balanceado de instancias

Al igual que en el problema de clasificación, en el de regresión **las instancias siguen estando muy desbalanceadas**, ya que como debemos predecir el *intermediate reward* de unos ticks futuros conociendo la información del presente, sabemos que **en la mayoría de los ticks dicho atributo no cambiará**.

Por tanto, hemos analizado en qué momentos la *intermediate reward* puede aumentar:

- **Cuando Mario recoge una moneda** que tiene en frente o salta para alcanzarla. (En el caso de nuestros bot, tanto el creado en el *Tutorial 1* como en esta práctica el segundo caso nunca se da, porque el objetivo de Mario es pasarse los niveles; por tanto, sólo consideraremos el primero).
- **Cuando Mario mata a un enemigo**, ya sea por un pisotón o tras usar un caparazón koopa (al igual que en el caso anterior, nuestros bot siempre matan a los enemigos mediante pisotones, por lo que los otros dos casos no los tendremos en cuenta).
- **Cuando Mario recoge otros objetos**, tales como el champiñón que da vida como las flores de fuego. Estos casos no los consideramos tampoco, ya que nuestros bot no se paran a recogerlos.
- **Cuando Mario gana el nivel**, que es otro caso que no podemos considerar, ya que grabamos en las instancias con un cierto desfase de ticks, por lo que esta instancia nunca se llega a grabar.

Los casos en los que puede disminuir son los siguientes:

- **Cuando Mario es alcanzado por algún enemigo**, ya sea un Goomba o una planta piraña que sale de la tubería (no hay más enemigos en el primer nivel de dificultad).
- **Cuando Mario pierde la partida**, que, como en el caso análogo de ganarla, no la podemos considerar porque tampoco se llega a grabar nunca.

De acuerdo con esta información, las situaciones que consideraremos a recoger en nuestro fichero de entrenamiento son las siguientes:

- **Mario tiene una moneda justo delante y se está moviendo hacia ella** (usando *move*), por lo que en el siguiente tick la recogerá.
- **Mario tiene un enemigo justo debajo o debajo y una celda a la derecha**, por lo que en el siguiente tick lo puede matar con un pisotón.
- **Mario tiene un enemigo en la diagonal arriba a la derecha o en frente y se está moviendo hacia él** (el movimiento que ejecute Mario aquí no es relevante, ya que la mayoría de las veces se choca con enemigos tras saltar o tras caer de un salto).
- **Mario no tiene una moneda justo delante**, por lo que no puede recoger ninguna.
- **Mario no tiene un enemigo ni debajo ni en frente**, por lo que no puede ni matarlo ni chocarse con él.

Así, y al igual que en el caso de clasificación, **filtraremos las instancias para guardar sólo las cinco situaciones que hemos indicado anteriormente**, de modo que también estén balanceadas entre ellas; ya que las dos últimas son los casos mayoritarios (en los que *intermediate reward* no cambia). Con el método de contadores que usamos en la parte anterior, nos aseguraremos de que guardamos el mismo número de instancias de cada tipo.

3.2. Descripción de los atributos seleccionados

Para seleccionar los atributos para esta tarea de regresión, hemos considerado aquellos **atributos que necesitamos para describir las situaciones del balanceo de instancias**, además de incluir otros que también puedan ser informativos. Estos son:

- Las **celdas más inmediatas a Mario de *mergedObs***, es decir, las mismas que consideramos para clasificación. A pesar de que estos atributos son los menos informados que proveemos a Mario, debido a que el algoritmo no encontrará relaciones entre las celdas por ser aprendizaje atributo-valor, no los hemos desechado por completo.
- ***distanceEnemy* nos indica la distancia euclídea al enemigo más cercano**. A pesar de que en la parte anterior nos demostró ser un atributo menos informado que *isEnemy*, no queremos rechazarlo del todo.
- ***distanceCoin* es el análogo de *distanceEnemy*, pero para la moneda más cercana**. Así, con estos dos atributos cubrimos las dos fuentes que existen para modificar el *intermediateReward*.
- ***isOnGround* sirve para reconocer si Mario se encuentra en el suelo**. Creemos que este dato es importante ya que Mario no puede eliminar ningún enemigo mediante pisotón si se encuentra en el suelo.
- ***isDanger* tomará el valor *Enemy* en el caso en el que se encuentre un enemigo en las casillas [8][10] y [10][10]**, es decir, en diagonal arriba y debajo enfrente de Mario. En el caso contrario, tomará el valor *NoEnemy*. Esto nos indica situaciones en las que Mario corre peligro de ser arrollado por un enemigo.
- ***isEnemyBelow* indica si hay un enemigo situado debajo de Mario**. Pensamos que este será muy relevante ya que inmediatamente después de tener un enemigo debajo lo aplastará y ganará puntos.

- ***isCoin* indica si existe una moneda en la casilla que está delante de Mario**. Esto resulta útil ya que al recoger esa moneda Mario aumentará su puntuación.
- ***action* indica la acción llevada a cabo en el tick actual**. Esto no podía hacerse en el apartado de clasificación, pero ahora que debemos predecir la recompensa intermedia puede resultar útil conocer la acción que esta realizando Mario.
- ***reward* indica la recompensa intermedia en el tick que se está ejecutando**. Añadimos la recompensa ya que las monedas y los enemigos dan una puntuación fija, +16 y +10, respectivamente. Consideramos la recompensa de la que se parte para poder predecir aquella a la a que se va a llegar en un futuro como un **atributo fundamental**.

Para la tarea de regresión hemos considerado **otros cuatro algoritmos distintos** para analizar su rendimiento para diferentes números de ticks: (***n+6*, *n+12*, *n+24***). Estos son: **M5, IbK1, IbK3 e IbK7**; siendo M5 e Ibk los únicos algoritmos que hemos aprendido en clase para regresión. Vamos a probar distintos valores para los vecinos más cercanos en Ibk, para ver cuál de ellos es que el que funciona mejor en nuestro problema. Asimismo, para cada tarea de regresión, analizaremos dichos algoritmos para tres ficheros de datos distintos:

- El **bot** generado a partir de **técnicas de aprendizaje automático** de la Parte 1 de esta práctica.
- El **bot procedural** creado en el *Tutorial 1*.
- El **agente** controlado por el **humano**.

Para esta parte, **usaremos tres parámetros en lugar de uno para valorar los algoritmos**: coeficiente de correlación, y dos errores relativos (ya que los errores absolutos pueden dar cifras engañosas): error absoluto relativo, raíz cuadrada del error cuadrático medio.

3.2.1. Evaluación para *n+6* ticks

Resultados para el bot de la Parte 1

Llevamos a cabo un experimento[1] usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 1700 instancias en cada hoja, ya que tenemos en total **33.774 instancias** en el fichero *entrenamiento_regression_6ticks_1.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido mediante la ejecución del *script P1Test_regression+6_1*, que consiste en ejecutar el fichero *P1BotAgent_pureBot_regression* 1000 veces con semillas de la 0 a la 999.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Parámetros	M5	IbK1	IbK3	IbK5
Coeficiente de correlación	0,9987	0,9561	0,9618	0,9598
Error absoluto relativo	3,9647 %	16,6339 %	16,9908 %	17,8562 %
Raíz cuadrada del error cuadrático medio	5,1414 %	29,3627 %	28,7162 %	28,6984 %

Cuadro 10: Resultados del análisis del experimento para *entrenamiento_regression_6ticks_1.arff*

[1] Debido a que hemos tenido problemas con el *Experimenter* para la parte de regresión, hemos llevado a cabo las mismas pruebas pero usando *Explorer*.

Como podemos ver, **los resultados de M5 son mucho mejores que los de cualquiera de los tres IbK**, alcanzando un Coeficiente de correlación muy alto (casi cercano al 100 %) y unos errores bastante bajos en comparación con los resultados de IbK (un 3,9 % comparado con errores de más de un 16 % para el error absoluto relativo y un 5,1 % comparado con errores de más del 28 % para la raíz cuadrada del error cuadrático medio). Además, podemos observar que **el mejor de los algoritmos IbK es el segundo** (parámetro 3) teniendo en cuenta los porcentajes de error, ya que la distancia no es muy pequeña (parámetro 1), ni muy grande (parámetro 5), aunque no hay apenas diferencias entre ellos.

De todos modos, **los mejores resultados de estas tres fases de experimentación serán los de esta primera (*n+6*)**, ya que es mucho más fácil predecir para distancias en el tiempo más cercanas. Asimismo, **los agentes bot consigan muy probablemente mejores resultados que el humano**. Y finalmente, aunque M5 parezca la mejor opción de todos los algoritmos seleccionados, veremos cómo se desarrolla en las siguientes pruebas.

Resultados para el bot del *Tutorial 1*

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 2000 instancias en cada hoja, ya que tenemos en total **39.690 instancias** en el fichero *entrenamiento_regression_6ticks_2.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido mediante la ejecución del *script P1Test_regression+6_2*, que consiste en ejecutar el fichero *P1BotAgent_original_regression* 1000 veces con semillas de la 0 a la 999.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Parámetros	M5	IbK1	IbK3	IbK5
Coeficiente de correlación	0,9988	0,9734	0,974	0,9708
Error absoluto relativo	3,6638 %	10,4285 %	11,7973 %	10,4285 %
Raíz cuadrada del error cuadrático medio	4,901 %	22,9318 %	22,8618 %	22,9318 %

Cuadro 11: Resultados del análisis del experimento para *entrenamiento_regression_6ticks_2.arff*

Como podemos ver, **los resultados de M5 son mucho mejores que los de cualquiera de los tres IbK**, alcanzando un Coeficiente de correlación muy alto (casi cercano al 100 %) y unos errores bastante bajos en comparación con los resultados de IbK (un 3,6 % comparado con errores de más de un 10 % para el error absoluto relativo y un 4,9 % comparado con errores de más del 22 % para la raíz cuadrada del error cuadrático medio). Además, podemos observar que **apenas hay diferencias entre los resultados de los tres IbK**, como en el caso anterior.

Comparando los resultados del otro bot y de este, **observamos una mejoría general de los resultados para los tres algoritmos IbK**, siendo la de M5 mucho más ligera, quizás debido a que tenía poco rango de mejora. La única diferencia significa entre este bot y el otro es que sus saltos son mucho más pequeños, aunque no tenemos claro si esa es la razón de sus mejores resultados.

Resultados para el humano

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 120 instancias en cada hoja, ya que tenemos en total **2.362 instancias**

en el fichero *entrenamiento_regression_6ticks_3.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido ejecutando manualmente el fichero *P1Human Agent_original_regression* 50 veces con semillas de la 50 a la 99; la mitad que en clasificación porque **recoge más instancias haciendo la mitad de iteraciones** y las pruebas del humano consumen mucho tiempo.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Parámetros	**M5**	**IbK1**	**IbK3**	**IbK5**
Coeficiente de correlación	0,998	0,8764	0,8591	0,8575
Error absoluto relativo	3,6339 %	28,1709 %	38,0194 %	39,8594 %
Raíz cuadrada del error cuadrático medio	6,3009 %	48,837 %	51,3017 %	52,055 %

Cuadro 12: Resultados del análisis del experimento para *entrenamiento_regression_6ticks_3.arff*

Sorprendentemente, **los resultados de M5 son casi igual de buenos para el humano que para los dos bot anteriores**, alcanzando un Coeficiente de correlación muy alto (casi cercano al 100 %) y unos errores muy similares. Sin embargo, **en el caso de IbK, los resultados empeoran mucho**, debido a que el *instance based learning* es difícil de aplicar cuando en una misma situación no se llevan a cabo las mismas acciones, como sucede con el humano. Además, podemos observar que **apenas hay diferencias entre los resultados de los tres IbK**, como en el caso anterior, con la diferencia de que empeora según aumenta el número de vecinos que tenemos en cuenta.

Teniendo en cuenta los resultados, vemos que **el mejor algoritmo de los propuestos es**, sin duda alguna, **M5** y que **el fichero que obtiene los mejores resultados es el bot del *Tutorial 1***, consiguiendo incluso resultados buenos con IbK.

3.2.2. Evaluación para *n+12* ticks

Resultados para el bot de la Parte 1

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 1700 instancias en cada hoja, ya que tenemos en total **33.730 instancias** en el fichero *entrenamiento_regression_12ticks_1.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido mediante la ejecución del *script P1Test_ regression+12_1*, que consiste en ejecutar el fichero *P1BotAgent_pureBot_regression12* 1000 veces con semillas de la 0 a la 999.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Parámetros	**M5**	**IbK1**	**IbK3**	**IbK5**
Coeficiente de correlación	0,998	0,9541	0,9602	0,9591
Error absoluto relativo	4,7251 %	17,3145 %	17,6509 %	184518 %
Raíz cuadrada del error cuadrático medio	6,3198 %	30,0387 %	28,1815 %	28,9568 %

Cuadro 13: Resultados del análisis del experimento para *entrenamiento_regression_12ticks_1.arff*

Aunque pensamos que los resultados de la predicción del tick *n+12* serían bastante peores que los del tick *n+6*, **han empeorando sólo ligeramente** para los tres parámetros que estamos teniendo en cuenta.

De este modo, **M5 se sigue manteniendo como la mejor opción**, manteniendo buenos resultados. **No será hasta *n+24* cuando veamos un mayor porcentaje de error**, ya que son 12 ticks de diferencia con estas pruebas, mucho más que los 6 de diferencia que estamos manejando hasta ahora; sabiendo que cuanto más distante se encuentre el futuro a predecir, peor será la predicción.

Resultados para el bot del *Tutorial 1*

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 2000 instancias en cada hoja, ya que tenemos en total **38.143 instancias** en el fichero *entrenamiento_regression_12ticks_2.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido mediante la ejecución del *script P1Test_ regression+12_2*, que consiste en ejecutar el fichero *P1BotAgent_original_regression12* 1000 veces con semillas de la 0 a la 999.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Parámetros	M5	IbK1	IbK3	IbK5
Coeficiente de correlación	0,9969	0,9713	0,9711	0.9686
Error absoluto relativo	5,7079 %	12,1271 %	13,3709 %	14,451 %
Raíz cuadrada del error cuadrático medio	7,832 %	23,833 %	24,0484 %	25,2759 %

Cuadro 14: Resultados del análisis del experimento para *entrenamiento_regression_12ticks_2.arff*

Al igual que en el caso anterior, **la diferencia de error en las predicciones entre *n+6* y *n+12* no es muy grande**, sobre todo en los casos de IbK. Sin embargo, en el caso de M5 es más acusado que en el bot anterior, **obteniendo peores resultados que el otro bot** en este rango temporal.

Resultados para el humano

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 120 instancias en cada hoja, ya que tenemos en total **2.371 instancias** en el fichero *entrenamiento_regression_12ticks_3.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido ejecutando manualmente el fichero *P1Human Agent_original_regression12* 50 veces con semillas de la 50 a la 99; la mitad que en clasificación porque **recoge más instancias haciendo la mitad de iteraciones** y las pruebas del humano consumen mucho tiempo.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:
En el caso del humano, se puede observar como empeora el coeficiente de correlación para todos los casos, lo que se refleja con el **aumento de los errores**. Suponemos que esto se debe a que la predicción ha de ser hasta medio segundo en el futuro, lo cual está más lejano en el tiempo que la predicción anterior. Como se ha dicho anteriormente, **predecimos un comportamiento similar**

Parámetros	M5	IbK1	IbK3	IbK5
Coeficiente de correlación	0,9959	0,8824	0,8601	0,8471
Error absoluto relativo	5,8752 %	28,9106 %	37,6198 %	41,6666 %
Raíz cuadrada del error cuadrático medio	9,0438 %	47,7334 %	51,1812 %	53,9152 %

Cuadro 15: Resultados del análisis del experimento para *entrenamiento_regression_12ticks_3.arff*

con los datos de *n+24* ya que es normal que el error aumente al aumentar el espacio de predicción, lo que no quiere decir que el modelo sea necesariamente peor.

Independientemente del aumento del error, ***M5* sigue pareciendo el algoritmo más prometedor**; teniendo ahora mejores resultados el bot implementado en esta práctica que el del *Tutorial 1*.

3.2.3. Evaluación para *n+24* ticks

Resultados para el bot de la Parte 1

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 1700 instancias en cada hoja, ya que tenemos en total **32.919 instancias** en el fichero *entrenamiento_regression_24ticks_1.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido mediante la ejecución del *script P1Test_ regression+24_1*, que consiste en ejecutar el fichero *P1BotAgent_pureBot_regression24* 1000 veces con semillas de la 0 a la 999.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Parámetros	M5	IbK1	IbK3	IbK5
Coeficiente de correlación	0,9949	0,9501	0,9568	0,9564
Error absoluto relativo	8,1345 %	19,5386 %	17,6509 %	20,3658 %
Raíz cuadrada del error cuadrático medio	10,1274 %	31,3215 %	29,3211 %	29,8414 %

Cuadro 16: Resultados del análisis del experimento para *entrenamiento_regression_24ticks_1.arff*

En este caso ya sí que se ha visto **un empeoramiento mayor del porcentaje de error**, tal y como se predijo en *n+12*, adquiriendo M5 un nivel de error mucho mayor del que nos tenía acostumbrados. Aun así, si la tendencia sigue como hasta ahora, este será el bot que mejor prediga la *intermediate reward*.

Resultados para el bot del *Tutorial 1*

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 2000 instancias en cada hoja, ya que tenemos en total **38.808 instancias** en el fichero *entrenamiento_regression_24ticks_2.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido mediante la ejecución del *script P1Test_ regression+24_2*, que consiste en ejecutar el fichero *P1BotAgent_original_regression24* 1000 veces con semillas de la 0 a la 999.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Parámetros	**M5**	**IbK1**	**IbK3**	**IbK5**
Coeficiente de correlación	0,9968	0,9703	0,971	0,9684
Error absoluto relativo	5,825 %	12,1663 %	13,3426 %	14,4715 %
Raíz cuadrada del error cuadrático medio	7,9658 %	24,0781 %	24,0484 %	25,3298 %

Cuadro 17: Resultados del análisis del experimento para *entrenamiento_regression_24ticks_2.arff*

A diferencia del bot anterior, en este **los resultados apenas empeoran notablemente**, por lo que su árbol de regresión se convierte en un fuerte candidato a ser el elegido para ser implementado, a expensas del resultado del humano.

Resultados para el humano

Llevamos a cabo un experimento usando **validación cruzada** (con 10 *folds*) en los algoritmos mencionados anteriormente para poder calcular los resultados. En el caso de M5 elegimos hacer una pre-poda con un mínimo de 120 instancias en cada hoja, ya que tenemos en total **2.339 instancias** en el fichero *entrenamiento_regression_24ticks_3.arff*, y así conseguimos que haya al menos un 5 % de las instancias en cada hoja (para que la regla de decisión correspondiente sea algo significativa).

Los datos del fichero de entrenamiento se han obtenido ejecutando manualmente el fichero *P1Human Agent_original_regression24* 50 veces con semillas de la 50 a la 99; la mitad que en clasificación porque **recoge más instancias haciendo la mitad de iteraciones** y las pruebas del humano consumen mucho tiempo.

Los resultados teniendo en cuenta las instancias correctamente clasificadas han sido los siguientes:

Parámetros	**M5**	**IbK1**	**IbK3**	**IbK5**
Coeficiente de correlación	0,9945	0,8534	0,8328	0,8356
Error absoluto relativo	8,9018 %	33,5897 %	41,5268 %	43,521 %
Raíz cuadrada del error cuadrático medio	10,4702 %	53,0011 %	55,5957 %	55,7081 %

Cuadro 18: Resultados del análisis del experimento para *entrenamiento_regression_24ticks_3.arff*

Como predijimos con el humano para la regresión en *n+12* ticks, **el coeficiente de correlación se ha visto afectado decreciendo mientras que el resto de errores han aumentado**. Como ya explicamos, esto es una consecuencia misma de aumentar el espacio de predicción.

Teniendo en cuenta todos los resultados analizados, consideramos que parece ser que **el mejor árbol de regresión es el que ha generado el bot del *Tutorial 1***, seguido por el otro bot. Debido a falta de tiempo, implementaremos sólo el primero de ellos.

3.3. Justificación del algoritmo seleccionado

A diferencia del caso de clasificación, para la tarea de regresión no hay algoritmos que se encuentren empatados en los mejores resultados (como sucedía con J48 y PART), ya que en el caso de *n+24*, **los resultados de M5 se han mantenido muy superiores a las otras tres opciones de IbK, sobre todo cuanto más aumentaba la ventana temporal**.

3.4. Descripción de la implementación

Teniendo en cuenta los datos presentados en la sección anterior decidimos implementar **un agente** usando tanto el **modelo generado por el bot del *Tutorial 1*** en nuestro agente de la Parte 1 (*P1Agent*), el bot del *Tutorial 1* (*P1BotAgent*) y en el humano (*P1HumanAgent*).

Para implementar el modelo de regresión lo primero que debemos hacer, al igual que en la clasificación, es **asegurarnos de que las instancias estén balanceadas**. En este caso nos interesa predecir la recompensa intermedia de Mario, la cual crece con su puntuación y decrece cuando es herido. Los puntos generalmente vienen dados por recoger monedas y matar goombas. Debido a que avanza hacia delante sin retroceder ni bloquearse le será imposible recoger champiñones o flores de fuego. Con todo esto en mente, identificamos las **siguientes situaciones a equilibrar**:

- Mario tiene una moneda delante y se desplaza hacia ella, lo cual hace que su puntuación aumente en el futuro.
- Mario no tiene monedas delante.
- Tiene un enemigo en una de las celdas que está debajo suya(celdas [10][9] y [10][10]), por lo que lo aplastará en el futuro incrementando su puntuación.
- Mario tiene un enemigo delante o justo arriba a la derecha. Contra este enemigo puede chocar y eso haría que perdiera puntos.
- Mario no tiene un enemigo delante.

El método para balancear estas instancias se hizo igual que en la parte anterior, haciendo uso de **contadores: *front_coin_counter*, *no_coin_counter*, *above_enemy_counter*, *front_enemy_counter*, *no_enemy_counter***. Dado que en este caso se puede dar más de una de esas situaciones al mismo tiempo, dividimos los contadores en aquellos que revisaba si había o no monedas cerca con los que no. Así, creamos dos bloques de condicionales mutualmente exclusivos (uno para las situaciones que involucran monedas y otro para las que involucran enemigos), haciendo uso de *if-else*.

Además incluimos los siguientes atributos:

- **distanceEnemy**: la distancia euclídea al enemigo más cercano.
- **distanceCoin**: la distancia euclídea a la moneda más cercana.
- **isOnGround**: 1 si Mario está en el suelo y 0 en caso contrario.
- **isDanger**: Se trata de un booleano que tomará su valor verdadero si existe un enemigo en la casilla frontal superior a Mario de tal manera que el enemigo le caerá encima próximamente hiriéndolo. También examina la celda de enfrente de Mario ya que si hay un enemigo también existe un riesgo de colisión.
- **isEnemyBelow**: Comprueba si hay un enemigo debajo de Mario o si el enemigo está en la celda frontal inferior a Mario de tal manera que muy probablemente Mario caerá sobre el enemigo aplastándolo he incrementado su puntuación
- **isCoin**: Indica si hay una moneda justo delante de Mario.
- **action**: Es la acción realizada por Mario en este tick. Se trata de un dato relevante porque puede detectar casos de las partidas del humano, en las que por ejemplo no reacciona a tiempo para saltar frente a un enemigo y se choca contra él.
- **reward**: La recompensa intermedia actual es un dato indispensable para calcular la futura ya que puede tomar como referencia sus valores a lo largo de los ticks y ver cómo ha evolucionado con según que situaciones.

Finalmente, el modelo de regresión generado por *M5* que implementamos es el siguiente, que consta de un solo modelo lineal:

futureReward = 21.1444 · **A** - 0.022 · **distanceCoin** + 37.5066 · **B** + 0.9968 · **reward** + 32.0017

A = 1 si MergedObs[9][11] = 80,93,5,25,3,2,1,-60,-85,-24; 0 en otro caso.
B = 1 si isCoin = Coin; 0 en otro caso.

Como observación decir que el modelo ha preferido mirar si hay algún enemigo en frente basándose en *MergedObs* que en los atributos que teníamos preparado para saber si había algún enemigo cerca (*distanceEnemy*, *isDanger*, *isEnemyBelow*).

3.5. Evaluación de los resultados

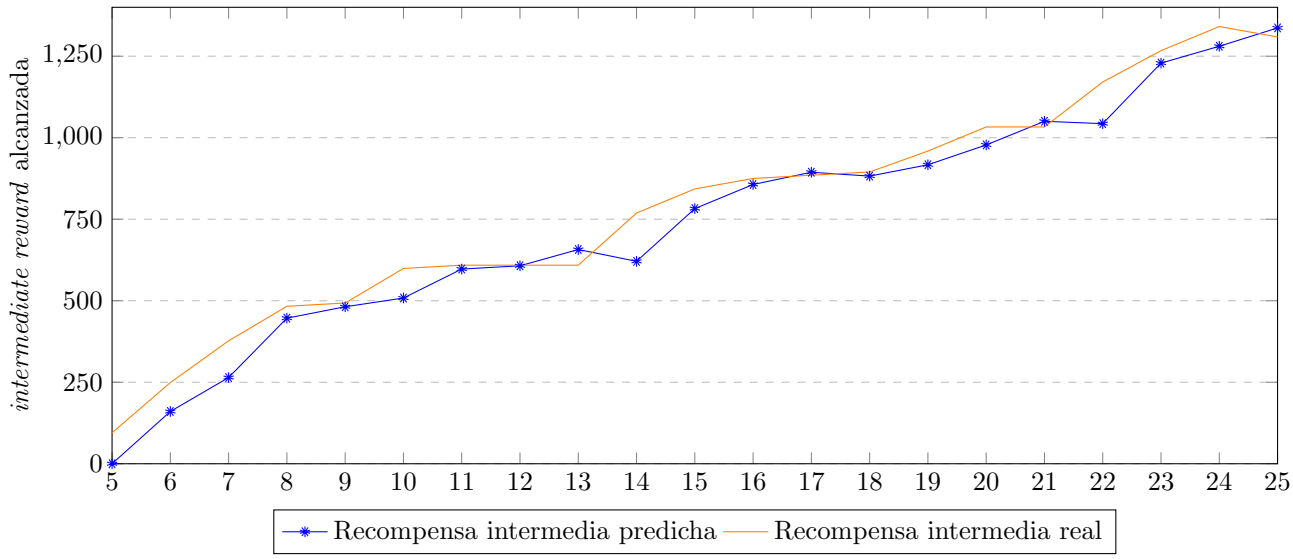

Figura 5: Evaluación del modelo de regresión en semillas nunca antes entrenadas

En esta gráfica se puede observar el rendimiento del modelo generado finalmente por **M5**. El modelo está siendo evaluado en la semilla 1001 (semilla **no** incluida en el conjunto de instancias de entrenamiento). Y el agente que controla a Mario es el *P1Agent*, es decir, el controlado por el modelo de clasificación generado en la primera parte de esta práctica.

La línea **naranja** representa el **valor real** de la recompensa intermedia para ese tick, mientras que la línea **azul**, es la **recompensa intermedia predicha** por el modelo.

En el **eje X** vemos los **segundos** (periodos de **24 ticks**) que empiezan en el 5 para evitar los primeros segundos donde no ocurre nada relevante para la recompensa intermedia (ni enemigos, ni monedas). Y, finalmente, en el **eje Y** podemos ver el valor que va tomando la **recompensa intermedia** a lo largo de la partida.

Aunque nos hubiera gustado realizar una evaluación mas exhaustiva, hemos decidido limitar el rango a 20 segundos porque de otra manera consumiríamos demasiado tiempo. De todos modos creemos que esta gráfica muestra perfectamente de lo que es capaz el modelo que generamos durante la realización de esta práctica.

4. Conclusiones generales

Tras la completar la práctica nos hemos dado cuenta de lo importante que es una buena selección de atributos. Además, pudimos ver como la creación de nuevos atributos fue crucial para mejorar los resultados de tanto la tarea de regresión y clasificación.

Analizando las gráficas obtenidas en cada modelo podemos concluir que estamos satisfechos con los resultados de nuestras implementaciones y que reflejan nuestra labor y esfuerzo en la extensa parte de experimentación.

5. Comentarios personales

La realización de esta práctica ha resultado muy útil en el aprendizaje de esta materia. Hemos podido ver como la selección de atributos correcta mejoraba notablemente los resultados del agente así como el impacto de añadir nuevos atributos.

Nos pareció realmente curioso que nuestro bot generado en esta práctica, que aprendió puramente de instancias seleccionadas del bot del Tutorial 3, haya resultado ser bastante mejor a la hora de completar niveles al decidir saltar siempre lo más alto posible.

Con respecto a la tarea de regresión, esta nos pareció más tediosa. Quizá porque no es tan visual como ver a Mario aprender que acciones realizar por si mismo y simplemente se reduce a predecir un valor numérico. Independientemente de esto, creemos que esta parte es necesaria para afianzar conocimientos teóricos estudiados en clase y poder ver en acción los algoritmos que se nos han enseñado.

En resumen, la práctica nos ha resultado muy adecuada y esperamos que las próximas nos ayuden a crear un agente mejor ya que es muy satisfactorio ver en acción el trabajo de experimentación realizado al implementar el modelo en el agente.