Prática 1: Clasificación y predicción

Grado en Ingeniería Informática

Aprendizaje Automático

Aitor Alonso Núñez NIA 100346169 Gr. 83
100346169@alumnos.uc3m.es
Daniel Gak Anagrov NIA 100318133 Gr. 83
100318133@alumnos.uc3m.es

Aitor Alonso Núñez NIA 100346169
Daniel Gak Anagrov NIA 100318133

uc3m | Universidad Carlos III de Madrid

Índice

1. Introducción **2**

2. Recogida de información **2**
2.1. Atributos seleccionados . . . 2
2.1.1. Atributos relacionados con la instancia del entrenamiento . . . 2
2.1.2. Atributos relacionados con la recompensa de mario . . . 2
2.1.3. Atributos relacionados con la matriz de observación . . . 3
2.1.4. Atributos booleanos o atributos binarios . . . 4
2.1.5. Otros atributos . . . 4
2.1.6. Atributo a clasificar . . . 4

3. Clasificación **5**
3.1. Algoritmos seleccionados . . . 5
3.1.1. J48 . . . 5
3.1.2. Ramdom Tree . . . 5
3.1.3. Random Forest . . . 5
3.2. Preporcesado realizado . . . 5
3.2.1. Borrado de atributos . . . 5
3.2.2. Combinación de datos de entrenamiento . . . 6
3.2.3. Balance de instancias . . . 6
3.2.4. Borrado de clases . . . 6
3.2.5. Otros preprocesados . . . 7
3.3. Análisis . . . 7
3.4. Conclusión . . . 10

4. Regresión **11**

5. Implementación **14**

6. Opiniones Personales **14**

Aitor Alonso Núñez NIA 100346169
Daniel Gak Anagrov NIA 100318133

Introducción

El siguiente documento contiene el registro de las decisiones y pasos tomados para la realización de la “Práctica 1: Clasificación y predicción”. De esta forma, dividiremos el documento en los siguientes apartados:

- **Recogida de información:** Detalla el proceso de la toma de datos de los agentes T3BotAgent y T3HumanAgent de MarioAI para su posterior procesamiento.
- **Clasificación:** En este apartado se detallan los algoritmos seleccionados para la creación de los modelos y el preprocesamiento realizado a los sets de datos para optimizar los resultados de nuestros clasificadores y eliminar el sesgo. Asimismo se presentan los mejores resultados conseguidos.
- **Regresión:** En este apartado se genera y se explica el modelo de regresión.
- **Implementación:** En esta ultima sección se detalla la implementación del clasificador y el modelo de regresión seleccionados.

Recogida de información

Atributos seleccionados

Con objetivo de hacer el mejor modelo posible, se ha buscado recolectar una gran cantidad de atributos para poder seleccionar los mas relevantes de ellos, siguiendo así la hipótesis del aprendizaje inductivo. De esta forma se han escogido atributos proporcionados por MarioAI, como derivaciones de operaciones entre atributos, todos explicados a continuación.

Atributos relacionados con la instancia del entrenamiento

- **timeSpent:** Tiempo gastado (int).
- **timeLeft:** Tiempo restante (int).

Atributos relacionados con la recompensa de mario

- **intermediateReward:** Recompensa ganada (int).
- **intermediateRewardWonLastTick:** Recompensa ganada en el último tick (int).
- **intermediateRewardWonLast6Ticks:** Recompensa ganada en los últimos 6 ticks (int).
- **intermediateRewardPredicted6:** Recompensa predicha en el tick n+6. (int)
- **intermediateRewardPredicted12:** Recompensa predicha en el tick n+12. (int)
- **intermediateRewardPredicted24:** Recompensa predicha en el tick n+24. (int)

Atributos relacionados con la matriz de observación

- **nearestEnemyLeftDistance:** Distancia euclídea al enemigo más cercano por la izquierda (x ∈ [0,9]) (double).
- **nearestEnemyLef_X:** Número de columna, coordenada x de la matriz de observación del enemigo más cercano por la izquierda (x ∈ [0,9]) (byte).
- **nearestEnemyLef_Y:** Número de fila, coordenada y matriz de observación del enemigo más cercano por la izquierda (x ∈ [0,9]) (byte)
- **nearestEnemyRightDistance:** Distancia euclídea al enemigo más cercano por la derecha (x ∈ [10,18]) (double)
- **nearestEnemyRight_X:** Número de columna, coordenada x de la matriz de observación del enemigo más cercano por la derecha (x ∈ [10,18]) (byte)
- **nearestEnemyRight_Y:** Número de fila, coordenada y de la matriz de observación del enemigo más cercano por la derecha (x ∈ [10,18]) (byte)
- **nearestBlockLeftDistance:** Distancia euclídea al bloque/ladrillo más cercano por la izquierda (x ∈ [0,9]) (double)
- **nearestBlockLef_X:** Número de columna, coordenada x de la matriz de observación del bloque/ladrillo más cercano por la izquierda (x ∈ [0,9]) (byte)
- **nearestBlockLef_Y:** Número de fila, coordenada y de la matriz de observación del bloque/ladrillo más cercano por la izquierda (x ∈ [0,9]) (byte)
- **nearestBlockRightDistance:** Distancia euclídea al bloque/ladrillo más cercano por la derecha (x ∈ [10,18]) (double)
- **nearestBlockRight_X:** Número de columna, coordenada x de la matriz de observación del bloque/ladrillo más cercano por la derecha (x ∈ [10,18]) (byte)
- **nearestBlockRight_Y:** Número de fila, coordenada y de la matriz de observación del bloque/ladrillo más cercano por la derecha (x ∈ [10,18]) (byte)
- **nearestCoinLeftDistance:** Distancia euclídea a la moneda más cercana por la izquierda (x ∈ [0,9]) (double)
- **nearestCoinLef_X:** Número de columna, coordenada x de la matriz de observación de la moneda más cercana por la izquierda (x ∈ [0,9]) (byte)
- **nearestCoinLef_Y:** Número de fila, coordenada y de la matriz de observación de la moneda más cercana por la izquierda (x ∈ [0,9]) (byte)
- **nearestCoinRightDistance:** Distancia euclídea a la moneda más cercana por la derecha (x ∈ [10,18]) (double)

- **nearestCoinRight_X:** Número de columna, coordenada x de la matriz de observación de la moneda más cercana por la derecha (x $\in$ [10, 18]) (byte)
- **nearestCoinRight_Y:** Número de fila, coordenada y de la matriz de observación a la moneda más cercana por la derecha (x $\in$ [10, 18]) (byte)

Atributos booleanos o atributos binarios

- **enemyNearRight:** Enemigo cercano por la derecha (en matriz de observación [8-9][10-11]) (bool)(0, 1)
- **blockNearRight:** Bloque cercano por la derecha (en matriz de observación [8-9][10-11]) (bool)(0, 1)
- **enemyAheadOnFloorHeight:** Enemigo delante a nivel del suelo (por debajo de los pies) (en matriz de observación [10][10-11]) (bool)(0, 1)
- **blockAheadOnFloorHeight:** Bloque delante a nivel del suelo (por debajo de los pies) (en matriz de observación [10][10-11]) (bool)(0, 1)
- **isMarioOnGround:** Hay foso/abismo/acantilado delante (en columna [10]) (bool)(0, 1)
- **isMarioAbleToJump:** (bool)(0, 1)
- **isMarioAbleToShot:** (bool)(0, 1)
- **isMarioCarrying:** (bool)(0, 1)
- **enemyWasKilledBin:** (bool)(0, 1)
- **marioWasInjuredBin:** (bool)(0, 1)
- **isSlopeDown:** (bool)(0, 1)

Otros atributos

- **coinsGainedLastTick:** Número de monedas recogidas en el tick actual (int)
- **marioMode:** marioMode (int)(0, 1, 2) == (Small, Large, Fire)
- **marioStatus:**marioStatus (int)(0, 1, 2) == (Small, Large, Fire)

Atributo a clasificar

- **actionKey:** Acción realizada o tecla pulsada (int)(0, 1, 2, 3, 4, 5, 6, 7, 8, 9) == (N, J, R, RJ, RS, RJS, L, LJ, LS, LJS)

Clasificación

Algoritmos seleccionados

Para la realización del modelo se han seleccionado diferentes algoritmos de cara a la experimentación con ayuda de weka. A continuación se describen los algoritmos probados:

J48

Es la implementación del C4.5 (una extensión del algoritmo ID3) realizada por el equipo de Weka. El objetivo del algoritmo J48 es la construcción de un árbol de desición binario capaz de clasificar nuevas instancias.

Ramdom Tree

Es método de construcción de árboles de decisión de weka que se forma a partir de un subset aleatorio de columnas.

Random Forest

Es una combinación de árboles de clasificación de forma que un nuevo input consiste en un vector de cada árbol del bosque, de forma que la elección del resultado, consiste en la mayoría de “votos” que realizan todos los árboles para clasificar esa nueva instancia. Se ha escogido este algoritmo únicamente para observar su comportamiento, no obstante, no se escogerá para la implementación ya que es costosa.

Preporcesado realizado

El procesado de datos varía levemente para los diferentes modelos generados, no obstante en este subapartado se van a explicar los principales preprocesados aplicados.

Borrado de atributos

Si únicamente el objetivo es llegar al final, existen muchos atributos en nuestro set de datos que son irrelevantes para el modelo a generar. Por ejemplo, la información de las posiciones de las monedas no influye en el objetivo de llegar al final. De esta forma se han seleccionado 14 que se utilizarán para crear los árbol de decisión. Estos atributos son los siguientes:

- **nearestEnemyRightDistance**
- **nearestEnemyRight_X**
- **nearestEnemyRight_Y**
- **nearestBlockRightDistance**
- **nearestBlockRight_X**

- **nearestBlockRight_Y**
- **enemyNearRight**
- **blockNearRight**
- **enemyAheadOnFloorHeight**
- **blockAheadOnFloorHeight**
- **isMarioOnGround**
- **isMarioAbleToJump**
- **isSlopeDown**
- **actionKey**

Combinación de datos de entrenamiento

Tras realizar los primeros modelos, se ha observado que los datos de entrenamiento de P1BotAgent tiene un comportamiento muy estructurado, mientras que los datos de P1HumanAgent son muy variantes, pues la descripción del comportamiento humano no es trivial. Esto dificulta la obtención de un clasificador correcto. Es por ello por lo que se ha optado por mezclar instancias de las dos clases.

Se ha elegido así un dataset para la experimentación con weka que contiene apoximadamente 5500 instancias generadas por el agente humano y 21000 generadas por el agente bot, consiguiendo así una relación cercana al 80 % bot - 20 % humano.

Balance de instancias

La generación de un árbol de decisión sobre una clase cuya la mayoría de instancias son de un tipo (en este caso del tipo R, pues el agente mayoritariamente avanzaba hacia la derecha en una partida), produce que arboles como los random trees con profundidad máxima, clasifiquen mayoritariamente nuevas instancias con esa clase. De esta forma se utilizan las funciones proporcionadas por weka, de forma que se duplican aquellas instancias minoritarias, y se borran instancias duplicadas de clases mayoritarias.

Borrado de clases

Al compensar las instancias en datos, clases que tienen relevancia, la pierden. Es decir, en nuestro caso el agente puede clasificar por N, J, R y RJ. Como hemos comentado en Balance de instancias, la clase R era mayoritaria, por lo que al balancear se ha conseguido que las cuatro clases tengan el mismo número de instancias, a un cuarto por cada clase. El problema es que de la toma de datos se tienen 4 instancias que acaban duplicándose para la clase N, eliminando información relevante de clases como J R y RJ, por tanto, algunos modelos están construidos sobre este atributo sin tener todas las clases.

Otros preprocesados

Se han aplicado otros preprocesados pero sin resultado visible de mejoría sobre la construcción del modelo, ya que por ejemplo, la normalización no influye en los algoritmos seleccionados, pero si que influye en el modelo de salida, porque el modelo se vuelve difícil de implementar ya que los ifs están sujetos a los datos normalizados, cosa que los datos de entrada en el MarioAI no están.

Análisis

El problema principal a la hora de evaluar estos modelos es que analíticamente no se puede saber que modelo va a tener mejor capacidad de predicción una vez sea implementado en MarioAI, se puede realizar una estimación por medio de la validación realizada a la hora de crear los modelos utilizando las matrices de confusión y el numero de estancias clasificadas correctamente, pero muchas veces, el resultado empírico no refleja las estimaciones. Es por ello por lo que el análisis consistirá en evaluaciones empíricas de las implementaciones de los modelos.

Como se ha comentado anteriormente, la elección de algoritmos, la determinación de sus parámetros y la selección de atributos se realiza mediante un proceso de retroalimentación. Esto quiere decir, que se intenta escoger estas variables, se genera un modelo y se prueba. Posteriormente se razona porque está funcionando mal, se realizan las modificaciones necesarias para intentar mejorarlo, y se vuelve a intentar hasta conseguir un porcentaje mínimo viable.

Para facilitar el proceso de retroalimentación mediante evaluaciones empíricas, se ha desarrollado un script (el cual ha sido entregado como `estadisticas.sh`) como prueba de rendimiento para evaluar los agentes implementados en MatrioAI. Esta evaluación dada un Agente y un número de niveles, ejecuta al agente en cada nivel, especificando el resultado del desempeño del agente en el nivel. Al finalizar, el script imprime un resumen con las siguientes datos:

- Total de niveles superados.
- Total de niveles no superados.
- Porcentaje medio de la superación de los niveles.
- Media de colisiones con enemigo por nivel.
- Media de monedas recogidas por nivel.
- Media de muertes de enemigos por nivel.

A pesar de que se han generado más modelos, se va a presentar el análisis de varios agentes que describen la evolución de la práctica, desde primeros agentes con los peores modelos, pasando por versiones intermedias, y terminando por el agente que ha conseguido mayor desempeño. Estos agentes son:

- **ART:** Agente que implementa un modelo mediante Random Tree con las siguientes características:
  - Uso de 14 atributos relevantes.

- La clase de salida puede tomar los cuatro valores: N, J, R, RJ.
- La clase de salida está equilibrada.

- **A48:** Agente que implementa un modelo mediante J48 con las siguientes características:
  - Uso de 14 atributos relevantes.
  - La clase de salida puede tomar los cuatro valores: N, J, R, RJ.
  - La clase de salida está equilibrada.
- **A48RRJ:** Agente que implementa un modelo mediante J48 con las siguientes características:
  - Uso de 14 atributos relevantes.
  - La clase de salida puede tomar dos valores: R, RJ.
  - La clase de salida está equilibrada.
- **A48JRRJ:** Agente que implementa un modelo mediante J48 con las siguientes características:
  - Uso de 14 atributos relevantes.
  - La clase de salida puede tomar tres valores: J, R, RJ.
  - La clase de salida está equilibrada.
- **ARTRRJ:** Agente que implementa un modelo mediante J48 con las siguientes características:
  - Uso de 14 atributos relevantes.
  - La clase de salida puede tomar tres valores: R, RJ.
  - La clase de salida está equilibrada.

Con estos agentes se realiza una evaluación empírica sobre 40 niveles consiguiendo los siguientes datos:

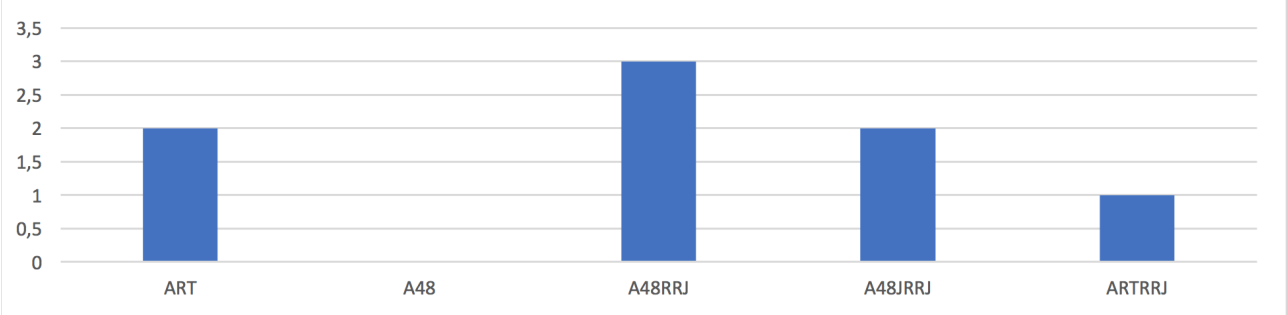

Figura 1: Total de niveles superados sobre 40

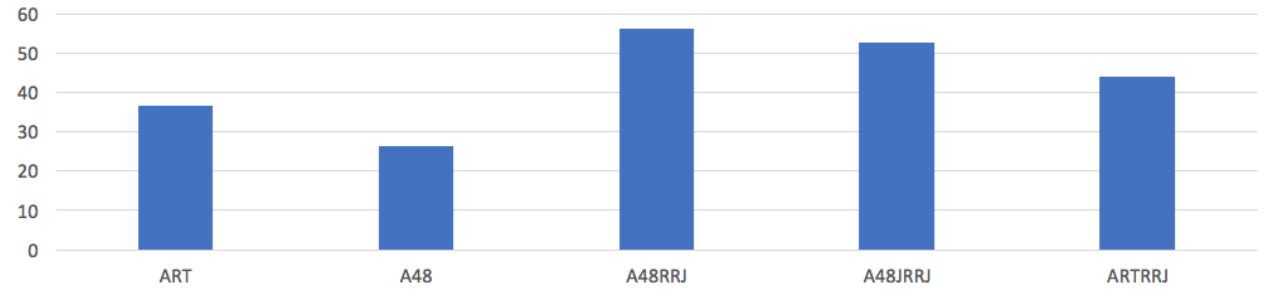

Figura 2: Porcentaje medio de la superación en cada nivel.

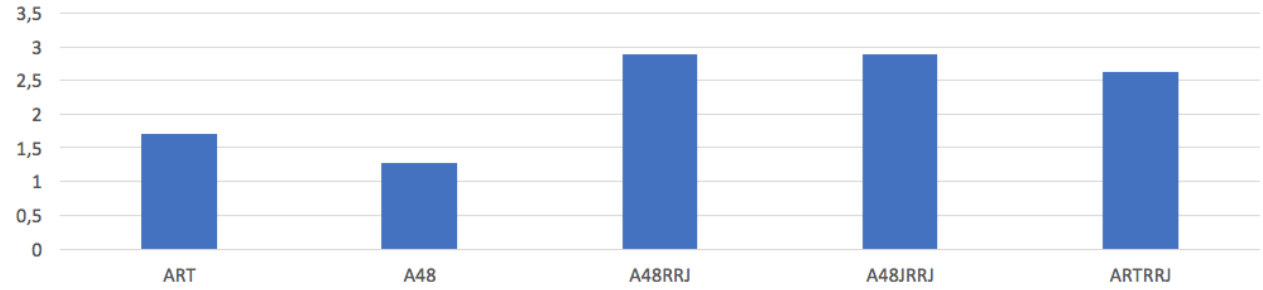

Figura 3: Media de colisiones con enemigo por nivel.

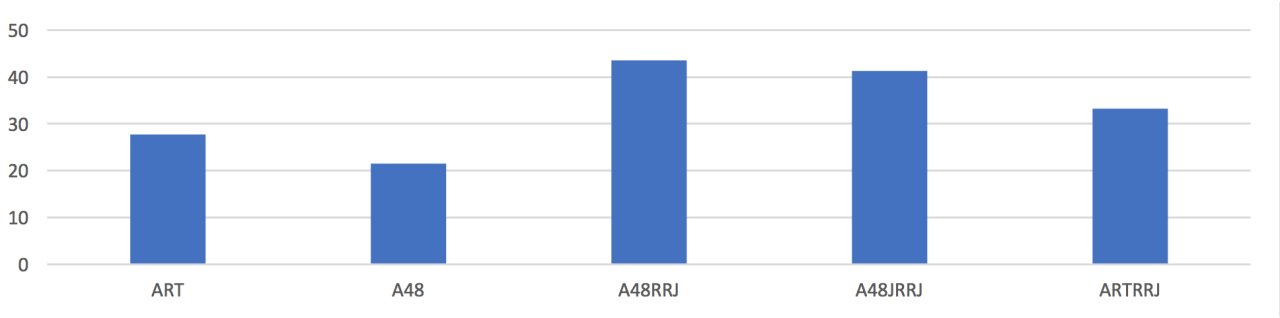

Figura 4: Media de monedas recogidas por nivel.

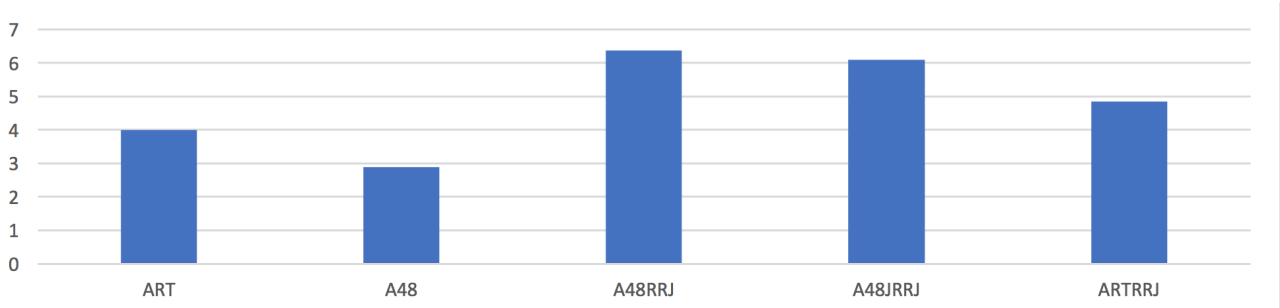

Figura 5: Media de muertes de enemigos por nivel.

Conclusión

A primera vista pareció que el ART tenía mejor desempeño, (incluso por su tendencia de no moverse en algunos niveles) pero al implementar el ARTRRJ, que también es un Random Tree pero con menos variables a clasificar, el resultado empeoró considerablemente. Tras unos ajustes, como se pude observar el A48RRJ es el modelo que mejor resultados ha conseguido, a pesar de sea el Agente que más colisiones posea con enemigos como se puede ver en la figura Media de colisiones con enemigo por nivel.. No obstante, al analizar la figura Media de colisiones con enemigo por nivel. con la figura Total de niveles superados sobre 40 podemos ver que Agentes con tasas menores de colisiones, tienen menos niveles superados. Esto se debe a que la razón de no conseguir superar el nivel no es la muerte del Agente, si no el agotamiento del tiempo debido a la dificultad encontrada para superar algunos .

Regresión

Para realizar la recta de regresión se cambia el objetivo, por lo que consecuentemente el preprocesado de clasificación no sirve para realizar un modelo de regresión.

Consecuentemente, el único preprocesado que se va a aplicar es la selección de atributos, resultando éstos en:

- **intermediateReward**
- **intermediateRewardWonLast6Ticks**
- **intermediaRewardPredicted12**
- **nearestEnemyLeftDistance**
- **nearestEnemyLeft_Y**
- **nearestEnemyRight_X**
- **nearestBlockLeftDistance**
- **nearestBlockLeft_X**
- **nearestBlockLeft_Y**
- **nearestBlockRightDistance**
- **nearestBlockRight_X**
- **nearestBlockRight_Y**
- **nearestCoinLeftDistance**
- **nearestCoinLeft_X**
- **nearestCoinLeft_Y**
- **nearestCoinRightDistance**
- **nearestCoinRight_X**
- **nearestCoinRight_Y**
- **enemyNearRight**
- **enemyAheadOnFloorHeight**
- **blockAheadOnFloorHeight**
- **isMarioAbleToJump**
- **actionKey**

- **intermediaRewardPredicted24**

De la misma forma en la que se se han barajado los modelos de clasificación, se prueban diferentes modelos de regresión. De forma que se escoge el modelo de regresión lineal para se ser implementado debido a su elevado coeficiente de correlación entre la clase a predecir y los atributos de entrada (0.998) para n + 24.

Para la evaluación empírica del modelo de regresión se ha implementado los tres modelos de regresión en el agente **P1BotAgentRegression**, generando datos de forma parecida a lo hecho en la sección de Análisis obteniendo como resultado los siguientes tres modelos de regresión:

*intermediaRewardPredicted24 = -0.282 * intermediateReward + -0.4332 * intermediateRewardWonLast6Ticks + 1.2849 * intermediaRewardPredicted12 + 0.1646 * nearestEnemyLeftDistance + -0.4056 * nearestEnemyLeft_Y + -1.0682 * nearestEnemyRightDistance + 0.7969 * nearestEnemyRight_X + -2.3371 * nearestBlockLeftDistance + -0.6307 * nearestBlockLeft_X + -1.315 * nearestBlockLeft_Y + -0.3964 * nearestBlockRightDistance + 0.5082 * nearestBlockRight_X + -2.9267 * nearestCoinLeftDistance + -0.3925 * nearestCoinLeft_X + -0.6596 * nearestCoinLeft_Y + -0.428 * nearestCoinRightDistance + 0.5277 * nearestCoinRight_X + -0.063 * nearestCoinRight_Y + -1.36 * enemyNearRight + 2.7432 * enemyAheadOnFloorHeight + -0.6076 * blockAheadOnFloorHeight + -2.2931 * isMarioAbleToJump + 4.3648 * enemyWasKilledBin + -1.4951 * actionKey + 62.1584*

*intermediaRewardPredicted12 = -0.5811 * intermediateReward + -0.175 * intermediateRewardWonLastTick + -0.3572 * intermediateRewardWonLast6Ticks + 1.2906 * intermediaRewardPredicted6 + 0.29 * intermediaRewardPredicted24 + -0.0693 * nearestEnemyLeftDistance + -0.3781 * nearestEnemyLef_X + 0.1532 * nearestEnemyLeft_Y + -0.1737 * nearestEnemyRightDistance + 0.1261 * nearestEnemyRight_X + 0.1246 * nearestEnemyRight_Y + 0.8084 * nearestBlockLeftDistance + 0.6317 * nearestBlockLeft_X + 0.441 * nearestBlockLeft_Y + -0.0942 * nearestBlockRight_X + 0.5381 * nearestCoinLeftDistance + 0.4871 * nearestCoinLeft_X + 0.1769 * nearestCoinLeft_Y + 0.0948 * nearestCoinRightDistance + 0.9151 * enemyNearRight + -0.6406 * blockAheadOnFloorHeight + -1.2063 * isMarioOnGround + 0.5919 * isMarioAbleToJump + 0.8761 * coinsGainedLastTick + 0.538 * actionKey + -19.2973*

*intermediaRewardPredicted6 = 1 * intermediateReward + -0.0008 * intermediateRewardWonLastTick + 1.0001 * intermediateRewardWonLast6Ticks + 0.0001 * intermediaRewardPredicted12 + -0.0001 * intermediaRewardPredicted24 + -0.0058 * nearestEnemyLeftDistance + -0.0068 * nearestEnemyLef_X + -0.0039 * nearestEnemyLeft_Y + 0.014 * nearestEnemyRightDistance + -0.0106 * nearestEnemyRight_X + -0.0024 * nearestBlockLeftDistance + -0.0032 * nearestBlockLeft_X + -0.0035 * nearestBlockLeft_Y + 0.0044 * nearestBlockRightDistance + -0.0042 * nearestBlockRight_X + 0.0033 * nearestBlockRight_Y + -0.0007 * nearestCoinLeftDistance + -0.0013 * nearestCoinLeft_Y + 0.0012 * nearestCoinRightDistance + -0.0016 * nearestCoinRight_X + 0.0002 * nearestCoinRight_Y + 0.0307 * enemyNearRight + 0.013 * blockNearRight + 0.0167 * enemyAheadOnFloorHeight + -0.01 * blockAheadOnFloorHeight + -0.015 * isMarioOnGround + -0.0145 * isMarioAbleToJump + -0.0105 * enemyWasKilledBin + 0.0173 * coinsGainedLastTick + -0.0561 * actionKey + 0.384*

Tras su ejecución en MarioAi, hemos obtenido los siguientes resultados:

```
######## Resultados de la regresión: ########

#### Predicted n + 6 ####
De un total de 922 ticks ha predicho correctamente 13 ticks (0.0%) y 903 ticks
erróneamente (100.0%).
La acumulación de intermediateReward predicha ha sido 940128 y el valor real es
de 928074, lo que nos da una desviación de 12054

#### Predicted n + 12 ####
De un total de 922 ticks ha predicho correctamente 0 ticks (100.0%) y 910 ticks
erróneamente (0.0%).
La acumulación de intermediateReward predicha ha sido 951316 y el valor real es
de 928074, lo que nos da una desviación de 23242

#### Predicted n + 24 ####
De un total de 922 ticks ha predicho correctamente 0 ticks (100.0%) y 898 ticks
erróneamente (0.0%).
La acumulación de intermediateReward predicha ha sido 968758 y el valor real es
de 928074, lo que nos da una desviación de 40684
```

Implementación

Para la implementación del árbol decisión en MarioAI se ha utilizado la herramienta weka para el preporocesado de los datos y la generación resultante. Un árbol resultante de weka podría tener el siguiente aspecto:

```
blockNearRight = 0
|   nearestEnemyRight_X < 11.5 : 2 (1758/36)
|   nearestEnemyRight_X >= 11.5 : 2 (4541/35)
blockNearRight = 1
|   nearestEnemyRight_Y < 12.5 : 2 (763/353)
|   nearestEnemyRight_Y >= 12.5 : 2 (260/85)
```

Estos modelos son fácilmente traspasables a la plataforma MarioAI siempre que no tengan un excesivo tamaño. Fácilmente traspasable código java, siguiendo el ejemplo anterior.

```
if(this.blockNearRight == 0){
    if(this.nearestEnemyRight_X < 11.5){
        action[Mario.KEY_RIGHT] = true;
    }else{
        action[Mario.KEY_RIGHT] = true;
    }
}else{
    if(this.nearestEnemyRight_Y < 12.5){
        action[Mario.KEY_RIGHT] = true;
    }else{
        action[Mario.KEY_RIGHT] = true;
    }
}
```

Opiniones Personales

Esta práctica es una simple visualización de la aplicación de técnicas de Aprendizaje Automático. No obstante, la carga de trabajo sumada a las previas entregas hace que la asignatura requiera más horas deseadas, haciendo que la calidad de trabajo tenga que ser disminuida. No obstante, al ser una práctica tan visual, resulta muy desafiante conseguir buenos resultados, lo que la hace ligera y bien llevadera. De esta práctica, se debería de mejorar el apartado de predicción, pues para es conceptualmente diferente al resto de la práctica, y no ha quedado claro el enfoque que se debe de tomar en ese apartado.