Aprendizaje Automático

Práctica 1:
Clasificación y predicción

CESAR MANUEL ARROJO LARA 100383510
JOSE MANUEL FRIAS SALVACHUA 100383533

uc3m | Universidad Carlos III de Madrid

Indice

Fase 1: Toma de Datos **2**

Fase 2: Clasificación **4**

Resultados Tutorial 1: 5

Ahora tocaría ver con otros mapas 6

Resultados Keyboard: 7

Resultados Keyboard en otros mapas: 8

Fase 3:Regresión **9**

Resultados tutorial1 mismos mapas 9

Resultados tutorial1 otros mapas 10

Resultados keyboard mismos mapas 11

Resultados keyboard otros mapas 11

Fase 4: Construcción de un Agente Automático **12**

Respuestas a las preguntas del apartado 7 **14**

Conclusiones y dificultades **14**

Fase 1: Toma de Datos

Para poder iniciar la toma de datos para entrenar a nuestro agente fue necesario modificar el printLineData() que habíamos creado en anteriores entregas. Esta vez los atributos que guardamos son:

- Posición en X del Pacman
- Posición en Y del Pacman
- Un atributo booleano por cada acción del pacman e indicando true si es legal esa posición
- Atributos numéricos indicando la posición en cada uno de los ejes del fantasma más cercano
- El número de fantasmas vivos en el momento
- Atributo numérico indicando la distancia al fantasma más cercana
- Atributo numérico indicando el número de comida restante
- Atributo numérico indicando la distancia a la comida más cercana
- Posiciones en X e Y de la comida más cercana
- Atributo clase donde guardamos la acción que se ha cogido teniendo en cuenta el resto de atributos
- 4 atributos numéricos más indicando tanto el score actual como los futuros

Esta selección de atributos podemos dividirla en varias partes y es posible que a la hora de empezar a analizarlos en profundidad para la clasificación, varios de ellos desaparezcan, pero por ahora decidimos tener la mayor cantidad posible para evitar tener que volver a tomar instancias constantemente. Las distintas secciones de los atributos son:

- Pacman: Contiene su posición y los atributos reflejando las acciones legales en el momento. Lo más probable es que estos atributos no sean filtrados ya que el agente necesitará en todo momento saber dónde está y las acciones que puede hacer. Si quitasemos esto, el agente podría intentar tomar acciones ilegales haciendo que el programa fallase. Y su posición es clave ya que se usa para el cálculo de adonde tiene que ir cuando la comparamos con la del resto de elementos en pantalla
- Fantasmas: Posición del fantasma más cercano y su número restante. Quizás el número restante podría filtrarse ya que podemos ir comprobando si las casillas tienen guardado None en vez de una posición numérica pero de todas maneras existe la posibilidad de que agilice algunos cálculos. Las posiciones las vemos clave guardarlas ya que, para que el Pacman se los coma, necesitamos saber dónde están.
- Comida: Número de comida restante, Distancia a comida más cercana y posición a comida más cercana. En un primer momento se pensó en añadir atributos de posición por cada comida pero nos dimos cuenta de que es imposible saber este número antes de iniciar el proyecto ya que distintos mapas pueden tener distinta comida entonces correríamos el riesgo de, o no tener atributos suficientes para el mapa que estamos jugando en el momento, o tener un exceso de atributos que no sirvan para nada. Por lo que decidimos mantener sólo la comida más cercana que, en el fondo, es a la que el agente debería ir. El número de comida también es importante, porque 1 de comida equivale a 100 puntos por lo que, a menos que tengamos un mapa extremadamente grande en el que la ida y vuelta de la comida suponga una pérdida incalculable, suele convenir comerse toda la comida del mapa

y dejar este atributo a cero antes de terminar la partida. La distancia es un atributo que quizás podramos filtrar porque también se puede inferir de los atributos de posición pero a la vez, estos atributos de posición sólo nos señalarían la distancia manhattan y teniendo muros esto no tiene porqué indicar el mejor camino.

- Clase y scores: Atributos que indican que clase hemos elegido (qué acción ha tomado el pacman teniendo en cuenta los atributos) y el score que tiene en el momento y en los tres instantes a futuro. Para clasificación el score no nos sirve de nada así que los filtraremos pero será algo clave a la hora de hacer regresión.

El tomar los scores a tres ticks de distancia supuso un problema un poco más complicado de lo que al principio podíamos suponer. Decidimos usar un array como buffer en el que íbamos guardando los distintos gameStates que íbamos obteniendo a la vez que vamos guardando los scores en otro array. La idea es ir guardando todo para que cuando termine la partida se escriba todo en el archivo de texto y así podemos acceder a los scores "futuros". Esto generaba un problema que era que en el último estado el score que se guardaba era el de antes de comerse al último fantasma (por lo que no sumaba los doscientos puntos) y luego el problema de que el último estado no poseía tres scores en el futuro. Decidimos que para esto, antes de escribir en el archivo, guardaríamos 3 scores extra, todos con el mismo valor de haber ganado los doscientos puntos por comerse menos 1 de pasarse el turno. Para poder activar esto sólo cuando el juego terminase decidimos añadir un parámetro ending a la función printLineData() que es llamada desde el último agente al terminar el juego.

Con esto pasamos propiamente a la toma de datos. Los mapas que se usaron para ambos modelos de toma (con usuario y automático) fueron los mismos, por si decidíamos en algún momento testear de forma cruzada. Los mapas elegidos para el entrenamiento fueron:

- OpenHunt
- smallClassic
- contestClassic
- basicMap
- Pokeball (mapa personalizado que usamos en la primera entrega)

Cada mapa se corría 3 veces en random, 4 veces con los fantasmas parados para las instancias de entrenamiento y 2 veces cada modo en las de test. Decidimos añadir una vez más en las de entrenamiento de fantasmas parados ya que a la hora de jugar con usuario y fantasmas aleatorios el movimiento random rápido de los fantasmas hacía que fallásemos algunas veces o no tomásemos la mejor decisión. Decidimos estos mapas para tener una mezcla entre las dos posibilidades que son Hunt y Classic ya que no sabemos qué nos podríamos encontrar a la hora de recibir un mapa futuro. Los mapas classic tienen un problema que consiste en la reaparición de los fantasmas una vez comidos, lo que hace que haya bastantes problemas a la hora de tomar bien los atributos. Sin embargo, prescindir de ellos no era una opción viable

Otros mapas para testear:

- BigHunt
- SixHunt
- capsuleClassic
- newmap
- testClassic

A la hora de tomar los datos para el tutorial 1 decidimos invertir un poco más de tiempo para intentar que fuese lo más preciso posible a una partida buena en vez de intentar cosas a lo loco hasta terminar. Para esto ayudó bastante encontrar en el archivo Distance Calculator el Distancer que permitía saber la distancia entre dos puntos teniendo en cuenta los muros y similares como si de una aplicación del algoritmo de Floyd-Warshall se tratara. También decidimos cambiar el método probabilístico que habíamos diseñado (permitiendo que nos desobedeciera en caso de entrar en un bucle), con algo que fuese más al punto. Para esto decidimos usar el distancer que acabábamos de conocer. Decidimos seguir una política de conseguir el máximo de puntos posibles por lo que nuestro agente se centraría en ir a por ellos, pero también pensamos que quizás sería un pérdida de tiempo ignorar los fantasmas por el camino así que decidimos que si un fantasma se encontraba a menos de tres movimientos de distancia con una comida a más de 4 movimientos, pasaríamos a comernos al fantasma y luego continuariamos nuestro camino. Más tarde tuvimos que modificar esto para que sólo se comiese los fantasmas si quedaba otro ya que si se comía a todos los fantasmas el juego se acababa y no había forma de completar lo que se había planeado. Así pues, si el juego era normal mirábamos los resultados que tendría ejecutar cada una de las distintas acciones, mirando cuál reducía más la distancia a nuestro objetivos, y ejecutábamos esta acción. Este cálculo se hacía en referencia a la caza de fantasmas y de comida (de ahí las dos funciones).
El algoritmo tiene algunos problemas ya que durante la ejecución de distintas pruebas vimos que si el último fantasma está en el camino directo a la comida, el agente se lo comerá y el juego acabará sin realmente haber cumplido su cometido. Esto provoca ruido en algunas de las instancias ya que el atributo de de comida restante seguirá siendo mayor que 0 y de repente el atributo de fantasmas vivos sea ya 0 y el score tenga un pico. Pero no encontramos manera de resolver este comportamiento. Otro problema que se vio bastante presente en Capsule Classic es que si 2 fantasmas están completamente a la izquierda del mapa y van cambiando su posición constantemente, nuestro agente entra en un bucle de moverse de derecha a ziquierda del que le es imposible salir ya que primero valora las acciones en el eje x antes que el eje y. Intentamos hacerlo al revés pero encontrábamos el mismo problema en otros mapas. Una solución sería que en el caso de darse la misma distancia en los dos ejes, hacer un randomNumber para elegir a cara o cruz qué eje coger, pero este método también acabó resultando en problemas que no podíamos controlar. Aún así, y con estos fallos, nuestro agente pasaba de algo cuasi aleatorio a algo con un poco más de pensamiento

Fase 2: Clasificación

Las siguientes fases 2 y 3 han tenido que ser cambiadas y revisadas varias veces debido a problemas inusuales que aparecían y que nos obligaban a volver a la toma de datos con nuevos enfoques. Al final, lo que hay representado en esta memoria es la última iteración de todos esos cambios. Estos distintos enfoques serán comentados a lo largo de las representaciones gráficas.
Decidimos usar un conjunto de tablas para comparar los distintos resultados de una forma clara y precisa, ya que este método nos permite ver en un vistazo rápido, qué conjunto de atributos y método resultó el más eficaz y es fácil de comparar con el resto
Hemos decidido probar varios conjuntos:

- Todos atributos mencionados previamente. Lo denominaremos **SinScore**

- Eliminando la distancia a comida y fantasmas más cercanos y eliminando Stop: Stop siempre es posible así que no tiene sentido mantenerlo y la distancia podría no llegar a influenciar tanto (aunque sabemos que al no ser la Manhattan con la que tomamos decisiones, quizás no podamos inferirla del resto de datos). Lo denominaremos **SinDistancia**
- Eliminando el número de comida restante y de fantasmas vivos: Una de nuestras condiciones a la hora de decidir qué hacer tiene en cuenta el número de comida restante pero eso es algo que podemos inferir igualmente si con los datos de posición que ya tenemos, ya que tenemos un valor que representa esta posición inexistente. Lo mismo ocurre con los fantasmas así que de nuevo es algo que se podría inferir. Lo denominaremos **SinNúmero**
- Unión de los dos anteriores:Viendo que hemos quitado esos atributos ya que se pueden inferir, a lo mejor podríamos quitar tanto las distancias como los números para probar. Lo denominaremos **SinDis_Número**

La mayoría de clasificadores que vamos a probar son aquellos que ya hemos usado previamente en otros tutoriales, ya que entendemos su funcionamiento en cierta manera. A estos hemos decidido añadir algunos más:

- LMT: En la sección de clasificadores de árbol ya usábamos J48 y el intentar discretizar los datos para Id3 siempre obtuvimos problemas. Esto hizo que decidiesemos probar otro de los métodos disponibles y la idea de utilizar métodos logísticos en las hojas del árbol parecía una idea interesante
- PART: La idea de usar un enfoque de divide-and-conquer usando métodos similares a Id3 (C4.5) permitía otro acercamiento posible a este tipo de clasificador ya que el propio Id3 no nos funcionaba. Además, el hecho de sólo quedarse con las mejores hojas parecía prometedor
- RandomForest y Bagging: Aprendimos sobre estos métodos en clase y parecían eficientes a la hora de crear clasificadores con buenos resultados
- KStar: La idea de clasificadores que usaban la clase de la instancia más cercana parecía interesante. Además, el hecho de usar la entropía para medir esta distancia a las instancias cercanas sugería un avance sobre IBK

Resultados Tutorial 1:

	SinScore	**SinDistancia**	**SinNúmero**	**SinDis_Número**
J48	**91.6274 %**	91.0518 %	91.2611 %	89.7436 %
IB1	89.4296 %	89.2726 %	89.2726 %	89.5866 %
IB5	88.4354 %	89.011 %	87.5981 %	88.1737 %
IB3	89.325 %	89.7436 %	89.2726 %	89.325 %
NaiveBayes	33.0194 %	45.2643 %	24.5945 %	33.0194 %
PART	90.1099 %	89.9529 %	90.6855 %	89.9006 %
LMT	90.8425 %	90.1099 %	90.2669 %	89.4296 %

Random Forest	91.5751 %	91.3658 %	90.9995 %	90.6332 %
ZeroR	26.3736 %	26.3736 %	26.3736 %	26.3736 %
Bagging	91.1565 %	89.8482 %	90.9995 %	90.3192 %
KStar	90.6855 %	90.5808 %	89.9006 %	89.9529 %

Como podemos observar, el mejor clasificador acaba siendo J48 con todos los atributos, seguido de cerca por las puntuaciones de Random Forest y, un poco más alejado, PART. También podemos observar que al usar los mismos mapas, la clasificación suele ser bastante buena (quitando Naive Bayes y Zero R que siempre suelen salir bastante regular cuando tenemos tantas clases posibles).

Ahora tocaría ver con otros mapas

	SinScore	**SinDistancia**	**SinNúmero**	**SinDis_Número**
J48	40.4204 %	36.8168 %	44.6847 %	37.2372 %
IB1	44.4444 %	43.9039 %	51.6517 %	51.7117 %
IB5	41.5616 %	40.0601 %	50.2703 %	50.1502 %
IB3	45.5856 %	41.5015 %	48.5886 %	50.6306 %
NaiveBayes	28.4084 %	36.8168 %	28.048 %	33.3333 %
PART	46.006 %	39.3393 %	40.3604 %	43.3033 %
LMT	40.6607 %	33.6937 %	38.979 %	35.9159 %
Random Forest	46.5465 %	46.5465 %	**54.0541 %**	**54.4745 %**
ZeroR	16.8769 %	16.8769 %	16.8769 %	16.8769 %
Bagging	41.8018 %	38.5586 %	48.2883 %	45.7057 %
KStar	39.5796 %	39.2192 %	43.4234 %	44.0841 %

Vemos que a la hora de comparar con otros mapas, nuestra precisión baja muchísimo. Esto seguramente sea porque a nuestro modelo le cuesta generalizar. La primera vez que ejecutamos las clasificaciones esto números se desplomaron aún más ya que inicialmente sólo habíamos cogido mapas Hunt para entrenar lo que significaba que a la hora de testear con otros mapas nos veíamos obligados a introducir mapas Classic para los que nos estábamos preparados. Debido a esto entremezclamos ambos conjuntos de mapas, incluyendo en el entrenamiento algunos de los Classic, haciendo que estos números suban un poco. Quizás con un entrenamiento de varios mapas más, no solo 5, conseguiríamos algo más general.

Esta vez, el mejor conjunto es el que no tiene en cuenta ni la distancia ni el número, seguido de cerca por el que no tiene número; ambos usando Random Forest. Esto contrasta fuertemente con los resultados anteriores ya que el conjunto SinDis_Número no es especialmente bueno a la hora de interactuar con los mismos mapas. Sin embargo si miramos SinNúmero vemos que tiene puntuaciones bastantes cercanas a la mejor de la anterior tabla y a la mejor de esta. Así pues, este sería por ahora nuestro mejor candidato para darle al weka_AI, ya que aunque no es el mejor en ninguno, establace un control bastante decente en ambos.

Resultados Keyboard:

	SinScore	**SinDistancia**	**SinNúmero**	**SinDis_Número**
J48	65.0198 %	60.2767 %	62.253 %	61.2648 %
IB1	58.498 %	59.6838 %	59.8814 %	60.4743 %
IB5	56.5217 %	58.1028 %	58.498 %	62.253 %
IB3	54.9407 %	58.498 %	54.9407 %	58.1028 %
NaiveBayes	24.5059 %	25.4941 %	26.2846 %	29.4466 %
PART	58.8933 %	**67.9842 %**	57.1146 %	56.1265 %
LMT	58.1028 %	56.7194 %	58.6957 %	58.3004 %
Random Forest	65.415 %	66.0079 %	58.1028 %	55.9289 %
ZeroR	27.4704 %	27.4704 %	27.4704 %	27.4704 %
Bagging	52.5692 %	52.9644 %	54.3478 %	53.3597 %
KStar	51.3834 %	49.8024 %	49.0119 %	47.4308 %

Como podemos ver, los resultados son bastante peores que los conseguidos en el tutorial 1. La respuesta de por qué ocurre es sencilla. Tutorial 1 sigue una serie de instrucciones al milímetro y tiene muy poca capacidad de irse de la ruta planeada. Nosotros no somos especialmente buenos, cometemos errores y cambiamos de estrategia sin darnos cuenta(a veces a algunas peores). Nuestros conjuntos de test también arrastran estos problemas pero no de manera estructurada lo que acaba haciendo que la generalización sea algo bastante difícil ya que es muy probable que teniendo la misma situación hayamos hecho cosas distintas sin darnos cuenta.

Esta vez vemos que el conjunto con mejor puntuación ha sido sin Distancia usando PART seguido de Random Forest, alejándose bastante del resto de puntuaciones. También podemos ver que todos los atributos usando J48 tiene buenos resultados como ocurrió con los datos del tutorial 1. Esto puede ser debido a que los atributos que decidimos tomar eran atributos que nos parecía lógico que afectasen a la decisión del agente (sin tener en cuenta la posibilidad de inferir unos de otros como hemos comentado antes).

Resultados Keyboard en otros mapas:

	SinScore	**SinDistancia**	**SinNúmero**	**SinDis_Número**
J48	44.5283 %	33.3333 %	37.8995 %	35.1598 %
IB1	23.3962 %	38.1279 %	36.3014 %	36.3014 %
IB5	33.2075 %	37.4429 %	37.2146 %	38.1279 %
IB3	26.0377 %	37.2146 %	34.0183 %	33.5616 %
NaiveBayes	35.4717 %	45.2055 %	42.0091 %	**46.5753 %**
PART	25.283 %	33.3333 %	34.2466 %	39.726 %
LMT	34.3396 %	27.8539 %	37.4429 %	25.1142 %
Random Forest	29.0566 %	44.7489 %	41.5525 %	40.1826 %
ZeroR	20 %	25.1142 %	25.1142 %	25.1142 %
Bagging	41.5094 %	25.5708 %	23.7443 %	24.2009 %
KStar	28.6792 %	26.9406 %	23.2877 %	26.9406 %

Como ocurrió en la parte anterior, la puntuación con otros mapas vuelve a disminuir comparada con el uso de los mismos mapas. Esta vez los datos son más parecidos a los de tutorial 1 (siendo estos últimos un poco mejores) lo que puede indicar que a lo mejor esa cualidad de fallo humano no afecta tanto a la hora de intentar generalizar lo aprendido.
Curiosamente, la mejor generalización la da SinDis_número con Naive Bayes, un clasificador que en el resto de apartados nos había dado pésimos resultados en comparación con el resto.

Una vez vistos los resultados en todas sus posibles variantes tendríamos que elegir cuál es el mejor conjunto de atributos y qué clasificador deberíamos elegir en el caso de querer usarlo en un futuro con mapas que se nos presenten. Como hemos ido comentando, los resultados obtenidos por keyboard no son demasiado buenos (como se podía esperar) y a la hora de usarlo con nuevos mapas tendríamos muchos problemas ya que no se ha seguido ninguna estrategia precisa. Así pues nos tendríamos que quedar con los conjuntos obtenidos con el tutorial 1. Como comentamos en su parte, nuestra mejor opción es usar su versión sin número y usando, si es posible, el Random Forest como clasificador (si no, usar PART);ya que esto es lo que mejpr resultado nos ha acabado dando en general y teniendo en cuenta todas las circunstancias. Sigue sin ser perfecto, ya que su puntuación con otros mapas apenas crece del 50% de acierto pero es lo mejor que podemos ofrecer con los conjuntos que hemos obtenido.

Fase 3:Regresión

Ahora vamos a intentar predecir los scores a distintas distancias de tiempo. Para esto usaremos los mismos conjuntos de antes sólo que añadiendo el Score actual para nuestro set de entrenamiento y los scores a las distintas distancias a los sets de test.

Decidimos usar M5P porque consistía en una variación del método que habíamos aprendido en clase. Además, intentamos M5Rules, que sería otra variación, pero este nos dio resultados peores así que decidimos mantenerlo. Para el segundo método a usar estuvimos entre Linear regression y el Multilayer Perceptron, que según leímos, eran los más utilizados a la hora de hacer regresiones. De nuevo, elegimos el Perceptron por sus mejores resultados aunque preferimos no cambiar ninguno de sus parámetros base, ya que al hacerlo siempre recibíamos resultados peores.

A la hora de comparar las distintas predicciones decidimos usar el coeficiente de correlación que se establecía entre lo que se había predicho con nuestro set de entrenamiento y el auténtico resultado que aparecía a la distancia de tiempo correspondiente.

Resultados tutorial1 mismos mapas

	Predicción a 1 tiempo Perceptron	**Predicción a 2 tiempos Perceptron**	**Predicción a 3 tiempos Perceptron**	**Predicción a 1 tiempo M5P**	**Predicción a 2 tiempos M5P**	**Predicción a 3 tiempos M5P**
Todo	0.9462	0.9456	0.945	0.9755	0.975	0.9742
SinDis	0.9544	0.9541	0.9536	0.972	0.9716	0.971
SinNumero	0.9072	0.9062	0.9053	0.9659	0.9655	0.9652
SinDis_Numero	0.9166	0.9159	0.9154	0.9656	0.9652	0.965

Por ahora casi todo se acerca bastante a lo que esperábamos en un principio. Muchas veces de un tiempo al siguiente el score sólo baja un punto haciendo que predecirlo sea bastante fácil. Sin embargo, los mapas classic pueden afectar bastante a esta predicción ya que al estar plagados de comida puede que del score actual a dentro de 3 turnos, el score haya aumentado en 300 puntos en vez de disminuirse. Esto puede hacer que haya veces en las que tengamos problemas a la hora de hacer la predicción.

También vemos que hay veces que nuestras predicciones sufren algunos fallos curiosos como el que podamos predecir mejor el score a dos instancias que en el momento actual. Tras los distintos esfuerzos por intentar corregir esto, no hemos conseguido descifrar el por qué. La solución que se nos ocurre es que haya un problema con la generalización de los mapas lo que hace que a veces el score que predecimos esté un poco alejado del score a una distancia de tiempo. Esto hace que cuanto más nos alejamos del score teorético actual, más nos acerquemos al que hemos predicho haciendo que su coeficiente de correlación sea mayor a medida que que buscamos más en el futuro

Resultados tutorial1 otros mapas

	Predicción a 1 tiempo Perceptron	**Predicción a 2 tiempos Perceptron**	**Predicción a 3 tiempos Perceptron**	**Predicción a 1 tiempo M5P**	**Predicción a 2 tiempos M5P**	**Predicción a 3 tiempos M5P**
Todo	0.2678	0.291	0.3036	0.1626	0.1686	0.1759
SinDis	0.3287	0.332	0.3264	0.1507	0.158	0.1651
SinNumero	0.0877	0.0899	0.0896	-0.1338	-0.1274	-0.1259
SinDis_Numero	0.1626	0.1689	0.166	-0.1316	-0.1252	-0.1232

Aquí viene el gran problema que no ha habido forma de arreglar. Podemos entender que a la hora de elegir la mejor acción con otros mapas, le cueste. Pero la predicción de resultados debería ser bastante sencilla. Sin embargo llegamos al mismo problema comentado anteriormente: Parece ser que nuestro último agente actúa de manera muy específica y que no consigue generalizar lo que aprende. Esto hace que cuando recibe sets de atributos de mapas distintos, no consiga discernir realmente lo que está pasando haciendo que el resultado obtenido se aleje mucho de la realidad. Por eso hay veces que los resultados mejoran cuanto más se alejan en el tiempo.
Debido a esta parte fue cuando decidimos implementar distintas estrategias:

- La original incluía la posición de los fantasmas en todo momento haciendo que cuando se muriesen marcasen su posición como 9999 en ambos ejes. Además, sólo habíamos tenido en cuenta para el entrenamiento los mapas de Hunt haciendo que cuando testeamos los mapas de Classic fuese completamente imposible predecir nada, ya que no estaban preparados para esa cantidad de puntos en pequeños momentos de tiempo debido a la comida
- Al darnos cuenta de este error en los mapas decidimos cambiar los mapas de entrenamiento a los que hay escritos al inicio del documento. Esto hizo que las predicciones fueran un poco mejores, pero en general seguían siendo bastante pésimas. Además, las predicciones ahora con los mismos mapas caían bastante hasta las 0.7
- Por último decidimos guardar únicamente la posición del fantasma más cercano ya que esto evitaría la posible confusión creada por estas posiciones 9999. De nuevo, esto mejoró los resultados a los que aparecen ahora mismo en esta memoria pero como se puede ver sigue habiendo un problema que no conseguimos discernir. Se probó a eliminar las instancias con 9999 pero tampoco sirvió como solución.

Así pues, aún sabiendo que no es el resultado óptimo, es aquel con el que hemos podido trabajar de mejor manera

Resultados keyboard mismos mapas

	Predicción a 1 tiempo Perceptron	**Predicción a 2 tiempos Perceptron**	**Predicción a 3 tiempos Perceptron**	**Predicción a 1 tiempo M5P**	**Predicción a 2 tiempos M5P**	**Predicción a 3 tiempos M5P**
Todo	0.6322	0.6748	0.6322	0.3499	0.3742	0.3499
SinDis	0.7475	0.6822	0.6432	0.4135	0.3742	0.3499
SinNumero	0.6385	0.6039	0.5815	0.4583	0.4149	0.3882
SinDis_Numero	0.717	0.6534	0.6124	0.7316	0.6677	0.63

Con los resultados de regresión a teclado tenemos el mismo problema que teníamos en las clases sólo que ahora se ve un poco más pronunciado. Al no seguir un sistema constante debido a nuestros cambios humanos es muy complicado hacer un modelo de regresión que, teniendo en cuenta los atributos, de algo medianamente certero.
También podemos ver una diferencia considerable entre los resultados del Perceptron y M5P; datos que curiosamente se invierten cuando llegamos al último conjunto de atributos. Podríamos pensar que estos son los atributos que estaban dañando las predicciones de nuestro agente pero si miramos los resultados de este conjunto cuando los dato han sido tomados por el tutorial 1, vemos que simplemente es casualidad, ya que aquí, el resultado siempre es peor.

Resultados keyboard otros mapas

	Predicción a 1 tiempo Perceptron	**Predicción a 2 tiempos Perceptron**	**Predicción a 3 tiempos Perceptron**	**Predicción a 1 tiempo M5P**	**Predicción a 2 tiempos M5P**	**Predicción a 3 tiempos M5P**
Todo	0.5562	0.5522	0.546	0.8407	0.8331	0.824
SinDis	0.505	0.4959	0.4837	0.8407	0.8331	0.824
SinNumero	0.2778	0.2833	0.289	0.1794	0.1803	0.1813
SinDis_Numero	0.2599	0.2648	0.2713	0.8032	0.8025	0.7996

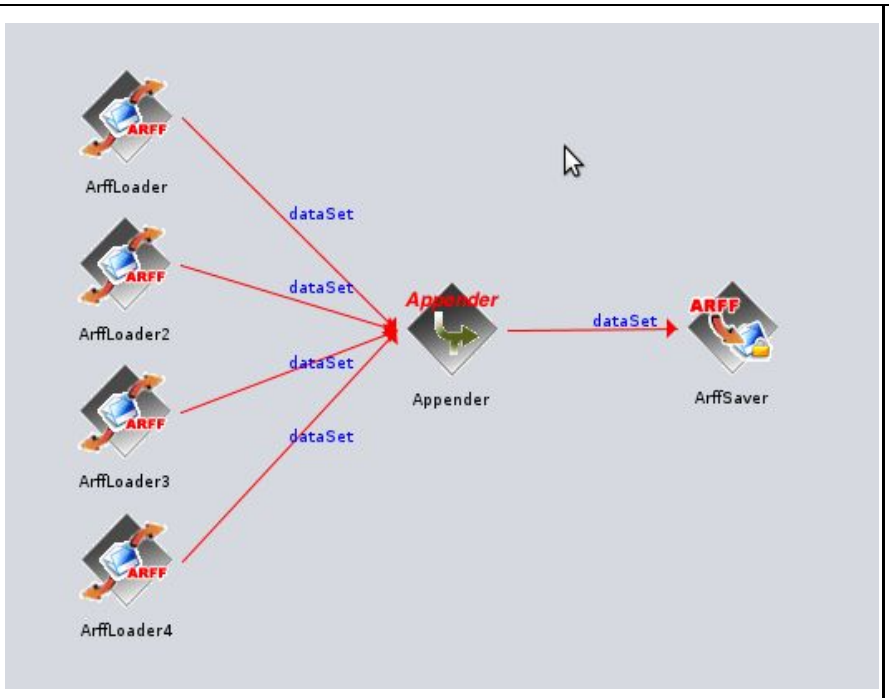
Aquí es cuando viene el elemento más curioso. Mientras que el Perceptron se comporta como esperaríamos (teniendo un poco menos de precisión que antes) el M5P se dispara haciéndose mucho mejor que su versión con los mismos mapas. Esto fue testeado varias veces y el resultado fue el mismo. Lo más probable es que sea una especie de casualidad pero podría indicar que la generalización que hace ahora funciona mucho mejor que su versión del tutorial 1. también podemos observar que cuando quitamos los atributos de

número, nuestra precisión cae en picado pero al balancearlo quitando también la distancia conseguimos que mejore.

Una vez comparado ya todo podríamos decir que nuestro mejor conjunto acaba siendo acaba siendo el de teclado sin los atributos de distancia usando M5P. Y con esto ocurre como ocurría con RandomForest en la fase de clasificación: No nos ha dado los mejores resultados siempre pero a la hora de generalizar es nuestra mejor opción así que si se le presentan nuevos mapas es el que mejor actuaría. Por su parte, los conjuntos de tutorial 1 no nos acaban sirviendo de mucho. Si seguimos usando los mismos mapas podíamos obtener mejores resultados pero a la hora de introducir cosas nuevas, no tendríamos una técnica de predicción para nada fiable. Por ello, lo mejor es el conjunto que hemos dicho.

Fase 4: Construcción de un Agente Automático

Para esta fase emplearemos los mismos algoritmos que para las anteriores. Para generar los modelos utilzamos un arff que, mediante el Knowledge Flow hemos construído mezclando los arff del tutorial1 y del keyboard, en los dos tipos de mapas.



test othermaps keyboard sinScore quitandoDistance.arff
+
test othermaps tutorial1 sinScore quitandoDistance.arff
+
test samemaps keyboard sinScore quitandoDistance.arff
+
test samemaps tutorial1 sinScore quitandoDistance.arff
=
test allmaps.arff

Además, hemos ejecutado con todos los modelos, cada uno de ellos generado independientemente en el Explorer de weka, sin resultados esperanzadores.

La siguiente tabla muestra el comportamiento del Agente que sigue las predicciones del modelo:

	Test_Allmaps.arff
J48	Se va a la esquina inferior izquierda ignorando los fantasmas
IB1	Se come a un fantasma y se va a la pared derecha
IB5	Se va a la esquina superior derecha ignorando los fantasmas
IB3	Se come la comida, se va a la pared derecha y muere en la esquina inferior derecha
NaiveBayes	Se va a la esquina inferior derecha ignorando los fantasmas
PART	Se va a la esquina inferior derecha ignorando los fantasmas
LMT	Se come la comida y se va a la pared derecha
Random Forest	**No se genera el modelo**
ZeroR	Ha predecido Este
Bagging	Se va a la esquina superior derecha ignorando los fantasmas
KStar	Se va a la pared derecha ignorando a los fantasmas

El modelo construído mediante aprendizaje es muchísimo más tonto que el programado. Simplemente es incapaz de jugar. No creemos que esté prediciendo mal la acción a tomar (según weka, la tasa de acierto es superior al 70% en todos ellos), pues todos los fantasmas están quietos, y el mapa es el por defecto. Quizás el error está en suministrarle demasiados pocos datos y muy genéricos.

Para intentar paliar estos resultados decidimos utilizar los arff del tutorial1, con los filtros que dan los mejores resultados. Eliminando el número de comida restante y de fantasmas vivos, guardando sólo las coordenadas del fantasma más cercano y manteniendo el resto de atributos. Mediante el KnowledgeFlow de weka, mezclamos lo siguiente:

test_samemaps_tutorial1_sinNUN.arff
\+ test_othermaps_tutorial1_sinNUN.arff
\+ training_tutorial1_sinNUN.arff
= test_allmaps_tutorial1_sinNUN.arff

Aún así, lo único que conseguimos con respecto a la anterior ejecución de los modelos es que J48 ahora se come la comida y a un fantasma. Pero luego muere contra el muro como los demás. Es por ello que no hemos guardado los .model de este último intento. No hay color entre el agente programado, que gana de forma eficiente, y estos agentes que ni siquiera ganan.

Respuestas a las preguntas del apartado 7

1. ¿Qué diferencias hay a la hora de aprender esos modelos con instancias provenientes de un agente controlado por un humano y uno automático?

La principal diferencia es la que hemos ido comentando a lo largo de la memoria que básicamente es la ejecución de una política de movimiento constante y una que tiene cambios y comete errores. Mientras el tutorial 1 es preciso en lo que hace, el agente humano falla y cuestiona si el camino que está haciendo es el mejor y duda.
La otra principal diferencia está en la acción STOP, los humanos necesitamos un tiempo en ticks para pensar y cambiar de acción, la más crucial es la de parar (que ocurre al principio de la partida y cuando te chocas), si una IA predice que se debe parar se va a quedar enganchado para siempre (pues si ni los fantasmas ni pacman se mueve, el estado no cambia). El agente programado nunca elige pararse, hasta que gana, nosotros nos paramos 2 de cada 10 ticks.

2. Si quisieras transformar la tarea de regresión en clasificación ¿Qué tendrías que hacer? ¿Cuál crees que podría ser la aplicación práctica de predecir la puntuación?

Simplemente habría que preparar el arff para que recibiera como parámetro el resultado de la regresión. El resto es simplemente elegir en weka uno de los clasificadores que acepten como clase un atributo numérico para intentar predecirlo y conseguir un modelo que nos acerque lo más posible a esos resultados. Poder predecir la puntuación facilitaría conseguir la solución óptima y permitiría saber la dificultad de un mapa sin jugarlo previamente.

3. ¿Qué ventajas puede aportar predecir la puntuación respecto a la clasificación de la acción? Justifica tu respuesta.

Para realizar un código que juegue automáticamente se necesitan muchas variables de entrada (posiciones, distancias, muros…). Si recibes una asociación de la puntuación que se estima conseguir con cada acción puedes elegir directamente la acción que maximice la puntuación. Básicamente dispones de una heurística gracias a la cual puedes aplicar algoritmos de búsqueda complejos, como A*.

4. ¿Crees que se podría conseguir alguna mejora en la clasificación incorporando un atributo que indicase si la puntuación en el instante actual ha descendido o ha bajado?

Sí, si se toman suficientes muestras, pues, tras un preprocesamiento permitiría purgar las “malas acciones” como por ejemplo dar vueltas o estamparse contra un muro.

Conclusiones y dificultades

La práctica está muy bien separada en 4 fases, pues poco tienen que ver una con la otra. En primer lugar se requiere código nuestro para la extracción de características y es posiblemente lo que más tiempo nos ha tomado. Conseguir extraer los datos desacompasados (los del tick actual con los de los 3 futuros) implica guardar información

del pasado en un buffer e imprimirla en grupos de 3 cada tick (además de simular las últimas cuando gana pues no hay más futuro).

Otro detalle de la primera fase es localizar el final, pues en cuanto pacman gana termina el programa sin volver a pasar por busterAgents. Finalmente decidimos hacer un apaño y guardar todo el output en un mismo string para imprimirlo todo al final. Para ello, desde game.py cuando termine el bucle principal, se llama a agent.printLineData() con un parámetro extra.

La fase 2 es la más interesante, porque requiere mucha expermientación conseguir un modelo que clasifique el 90% de los ejemplos correctamente. Quitamos y filtramos muchos datos, y cambiar de mapa se cargaba la eficacia completamente. Un dato quizás relevante es que no hay un algoritmo que sea bueno en todos los casos, pues los que predicen bien con muchos datos sin filtrar luego se la pegan en la regresión por ejemplo.

En la fase 4 el principal problema es instalar los prerrequisitos para javabridge. No nos ha resultado para nada trivial pues la instalación según el enunciado se queda corta. Por otra parte no hemos conseguido ningún resultado útil con nuestros agentes. Independientemente del algoritmo eran incapaces de jugar por lo que ha sido un desastre.

Los modelos obtenidos no parecen ser funcionales en esta aplicación, desarrollar un código para jugar óptimamente no es muy complicado y el número de parámetros necesarios de entrada es manejable. Quizás un agente automático es mejor elección cuando el problema de entrada es mucho más complicado de entender y procesar, tanto que el código programado es más pesado que una red de neuronas que, aunque acierte menos, se consigue antes. Si el problema es muy cambiante (como el layout) la IA que se ha adaptado a jugar en ciertas condiciones lo pasa mucho peor que un código preprogramado.

Por último, nos gustaría añadir que no tiene mucho sentido el intentar crear un agente automático que corra tanto los mapas Hunt como los Classic ya que sus reglas son distintas (la reaparición de fantasmas siendo la principal). Esto solamente ha servido para dar problemas y quebraderos de cabeza. Quizás si fuese más específico se podría haber llegado a un modelo capaz de conseguir algo. Pero una generalización de dos juegos distintos, aunque tengan algunas características similares, es simplemente una dificultad añadida sin motivo