Aprendizaje Automático

GRADO EN INGENIERÍA INFORMÁTICA

Práctica 1: Clasificación y predicción

Curso 2020/2021

Jorge Rodríguez Fraile, 100405951, Grupo 83, 100405951@alumnos.uc3m.es
Carlos Rubio Olivares, 100405834, Grupo 83, 100405834@alumnos.uc3m.es

Índice

Introducción

En esta práctica se desarrollará un agente automático de Pac-man a partir de la herramienta Weka. Dividiremos el proyecto en 4 fases. En la primera fase, se recogerán todas las instancias necesarias para la prueba y el entrenamiento, además de hacer un estudio de la relevancia de los atributos escogidos y cuales se pueden eliminar/añadir. La segunda fase es de clasificación del modelo donde se escogerán diferentes algoritmos y ficheros con diferentes atributos para generar el mejor clasificador posible. En el tercer paso, se centrará en predicción de clases con un método parecido al de la fase anterior, comparación de datos y algoritmos mediante gráficas. Por último, se comparará el agente que hemos obtenido con el de los anteriores tutoriales.

Fase 1: Recogida de la información

La función de extracción de datos proporciona en cada instancia, tanto información sobre el estado actual del juego, como del siguiente paso que se realizará. Los atributos que se han considerado de mayor utilizada son los siguientes:

- Posición de Pac-man en el estado actual en el eje x.
- Posición de Pac-man en el estado actual en el eje y, su posición es esencial en el caso de que busquemos situarlo cerca de los fantasmas y conocer si llega al límite del tablero.
- Fantasmas vivos en el estado actual esto nos permite saber cómo de cerca de terminar esta.
- La distancia al fantasma más cercano en el estado actual, eso nos permite ver cómo se va reduciendo según nos acercamos, orienta a Pac-man.
- La puntuación en el estado actual es una manera alternativa al número de fantasmas para saber que nos acercamos al fin, que nos hemos comido un fantasma.
- Posición de Pac-man en el estado siguiente en el eje x.
- Posición de Pac-man en el estado siguiente en el eje y.
- Fantasmas vivos en el estado siguiente.
- La distancia al fantasma más cercano en el siguiente estado.
- Puntuación en el estado siguiente.
- Por último, la clase, que es la acción de Pac-man que hará en el siguiente estado.

Estos datos nos han generado unos resultados que creíamos que se podrían mejorar, por lo que intentamos comenzar todo el proceso de nuevo y pensamos en atributos que tuvieran algo de relevancia en este problema, y acabamos con una lista de 25 atributos:

- **pacmanX**: Posición de Pac-man en el eje X
- **pacmanY**: Posición de Pac-man en el eje Y, como antes para ver cómo de cerca de los bordes o de los fantasmas está.
- **Width**: Anchura del mapa, para controlar que no se salga del mapa.
- **Height**: Altura del mapa, para controlar que no se salga del mapa.
- **gGLiving**: Estado del fantasma G (vivo o muerto), se tienen en cuenta 4 fantasmas.
- **livingGhosts**: Número de fantasmas vivos, es un atributo derivado del estado de los fantasmas.
- **gGX**: Posición X del fantasma G, se tienen en cuenta 4 fantasmas.

- **gGY**: Posición Y del fantasma G, se tienen en cuenta 4 fantasmas, eso junto a la posición de Pac-man podrían ayudar a acercarse de manera precisa.
- **gGD**: Distancia de Pac-man hasta el fantasma G, se tienen en cuenta 4 fantasmas, es un atributo derivado de la posición de Pac-man y la de cada uno de los fantasmas.
- **minGhost**: Distancia de Pac-man al fantasma más cercano, atributo derivado del anterior, que permite que se centre en la presa más cercana.
- **score**: puntuación actual.
- **score2**: puntuación del siguiente turno
- **class**: movimiento de Pac-man

Aun así, comprobando diferentes filtros, algoritmos y combinaciones de atributos los resultados no han sido mucho mejores que los obtenidos con los anteriores atributos, el mejor valor de predicción obtenido con estos atributos ha sido de 60.558%, y mejor valor con los atributos anteriores de un 59%, la diferencia es mínima y en el primer caso, los atributos son menos y son más 'limpios' por lo que se continuará con esta opción, aunque se adjunta el fichero de entrenamiento y de prueba para otros mapas de estos como training_keyboardN.arff y test_othermaps_keyboardN.arff respectivamente.

Los datos del siguiente paso los podemos obtener gracias a la función de obtener el sucesor de un gameState, getSuccessor.

Para el entrenamiento se han producido 458 instancias con el agente teclado, siguiendo las siguientes configuraciones tanto con RandomGhost como con StaticGhost:

training_keyboard.arff:

20Hunt
sixHunt
oneHunt
openHunt
smallHunt

Para probar el modelo producido con el entrenamiento anterior se han producido otras 230 instancias, con las siguientes configuraciones tanto con RandomGhost como con StaticGhost:

test_samemaps_keyboard.arff

smallHunt
openHunt
oneHunt
sixHunt

test othermaps keyboard.arff:

bigHunt
layoutAlumno
Newmap

training_tutorial1.arff:

smallHunt
oneHunt
openHunt
20Hunt
TrainHut1

Fase 2: Clasificación

Con los archivos de entrenamiento generados en la primera fase, se han generado distintos archivos derivados en los que se ha probado distintas combinaciones de atributos para evaluar cual es el mejor conjunto de atributos para el entrenamiento del modelo. Las modificaciones se han nombrado de la siguiente manera:

- **all**: Todos los atributos relativos al estado actual, que nos sirve de caso base para comparar con el resto de los conjuntos de los atributos.
- **sinLiving**: Todos los atributos del estado actual excepto el número de fantasmas vivos, que nos permitirá ver si afecta al algoritmo el número de fantasmas restantes o solo buscará acercarse a ellos, reduciendo la distancia.
- **sinMin**: Todos los atributos del estado actual menos la distancia más corta a un fantasma, por contrapartida del anterior este nos servirá para ver si lo que busca es que queden menos fantasmas en vez de acercarse a ellos que probaría que se los comiese.
- **sinPos**: Todos los atributos del estado actual menos la posición de Pac-man, este conjunto de atributos nos servirá para ver si no es necesaria la posición del jugador puesto que en el atributo de la distancia al fantasma más cercano se tiene en cuenta la misma.
- **sinScore**: Todos los atributos excepto la puntuación.

Este proceso se ha aplicado tanto a los datos de entrenamiento jugados por teclado o de manera autónoma (training_keyboard.arff y training_tutorial1.arff), como con los que son para probar los modelos (test_othermaps_keyboard.arff, test_samemaps_keyboard.arff, test_othermaps_tutorial1.arff y test_samemaps_tutorial1.arff). (10 archivos en total)

Además de los generados hasta ahora, se han creado otras dos variantes, una aplicando el filtro de normalizar los valores y otra de equilibrar el número de instancias de cada clase. (30 archivos en total)

- Se ha elegido el normalizado de datos para evitar que afecten las distintas dimensiones de los mapas, dado que se han probado mapas de multitud de tamaños, y las puntuaciones que toman gran variedad de valores.
- Se han balanceado las instancias para que tenga un conjunto equilibrado de cada clase y se pueda entrenar por igual.

Para cada uno de los archivos generados hemos evaluado el modelo que generaban para determinar cuál es el conjunto de atributos que genera mejor modelo.

Los hemos evaluado pasándolos por distintos algoritmos, que han sido: J48, IBk(1), IBk(3), IBk(5), IBk(7), PART, ZeroR, Naïve Bayes y KStar.

- **J48**, se ha empleado este algoritmo al ser una versión mejorada del que hemos estado estudiando en clase, ID3, y pensamos que la generalización puede ser acertada para predecir la dirección a la que se debe mover Pac-man.
- Se ha probado **IBk** (Instance Based Learner) con distintos valores para ver si al aumentar el número de vecinos que tiene en cuenta para predecir las instancias mejoraría los resultados.
- **PART**, al igual que con J48 lo hemos seleccionado por ser una derivación de un algoritmo del que conocemos su funcionamiento y como el problema no es muy grande no será problema el tamaño del árbol, que en este algoritmo no está optimizado.
- **KStar**, lo hemos seleccionado para ver si gracias a seleccionar instancias similares es capaz de imitar el comportamiento de una persona buscando a los fantasmas o el de nuestro algoritmo de BasicAgentAA.
- **ZeroR**, lo hemos seleccionado para observar cómo se comporta con nuestro problema y datos, pero no es un algoritmo del que esperamos obtener buenos resultados, dado que siempre se movería hacia la misma dirección, que no es lo óptimo.
- Por último, hemos utilizado **Naïve Bayes** como clasificador para utilizar otro de los algoritmos que conocemos y que no es de árbol de decisión, en este caso es un estimador de la clase, utiliza los valores de los atributos para predecir cuál será la clase más probable.

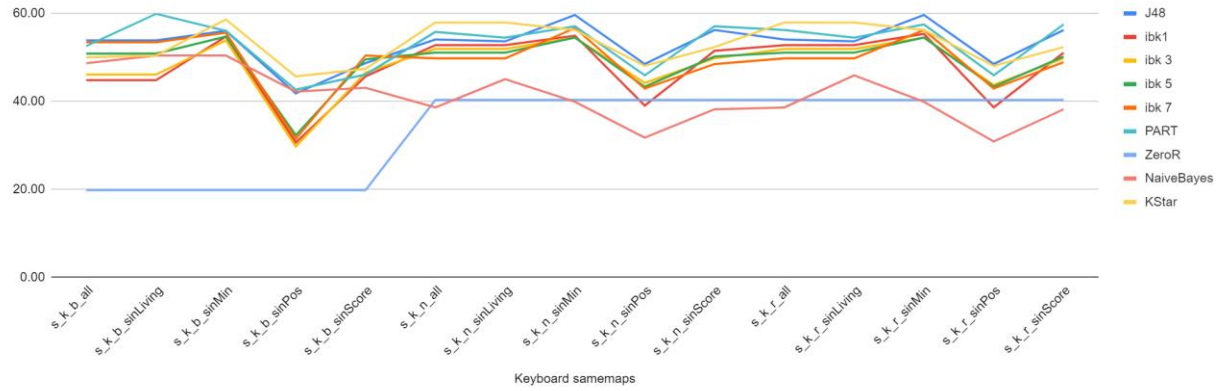
Para poder realizar esta evaluación se ha empleado el Experimenter. Con respecto a entrenar y evaluar los modelos con los datos correspondientes, se ha concatenado en el archivo de entrenamiento las instancias de prueba (samemaps o othermaps), procediendo de manera cuidadosa se ha calcula el porcentaje de instancias que son de entrenamiento para indicárselo a la herramienta y que proceda a evaluar los modelos. (30 archivos de samemaps y 30 de othermaps, 60 en total)

Para identificar los distintos archivos se ha seguido el siguiente patrón de nombrado de los conjuntos evaluados, mapa_agente_filtro_atrib.arff, en el que cada campo significa:

- **Mapa**: ‘o’ para othermaps y ‘s’ para samemaps.
- **Agente**: ‘k’ para teclado y ‘b’ para el BasicAgentAA.
- **Filtro**: ‘r’ es sin filtro, ‘n’ es normalizado y ‘b’ para clase balanceada.
- **Atrib**: Son los nombres descritos arriba.

A continuación, se presentan los datos obtenidos de la evaluación de los modelos medido en porcentaje de predicciones que aciertan la clase:

Keyboard samemaps	J48	IBk 1	IBk 3	IBk 5	IBk 7	PART	ZeroR	NaiveBayes	KStar
s_k_b_all	53.88	44.83	46.12	50.86	53.45	52.59	19.83	48.71	50.00
s_k_b_sinLiving	53.88	44.83	46.12	50.86	53.45	59.91	19.83	50.43	50.43
s_k_b_sinMin	56.03	54.74	53.88	54.74	55.60	56.03	19.83	50.43	58.62
s_k_b_sinPos	41.81	30.60	29.74	32.33	31.47	42.67	19.83	42.24	45.69
s_k_b_sinScore	48.71	45.69	46.55	49.57	50.43	46.12	19.83	43.10	47.41
s_k_n_all	54.08	52.79	51.93	51.07	49.79	55.79	40.34	38.63	57.94
s_k_n_sinLiving	53.65	52.79	51.93	51.07	49.79	54.51	40.34	45.06	57.94
s_k_n_sinMin	59.66	54.94	54.51	54.51	56.65	57.08	40.34	39.91	56.22
s_k_n_sinPos	48.50	39.06	44.21	43.35	42.92	45.92	40.34	31.76	48.07
s_k_n_sinScore	56.22	51.50	49.79	50.21	48.50	57.08	40.34	38.20	52.36
s_k_r_all	54.08	52.79	51.93	51.07	49.79	56.22	40.34	38.63	57.94
s_k_r_sinLiving	53.65	52.79	51.93	51.07	49.79	54.51	40.34	45.92	57.94
s_k_r_sinMin	59.66	55.36	54.51	54.51	56.22	57.51	40.34	39.91	56.22
s_k_r_sinPos	48.50	38.63	43.78	43.35	42.92	45.92	40.34	30.90	48.07
s_k_r_sinScore	56.22	51.07	49.79	50.21	48.93	57.51	40.34	38.20	52.36

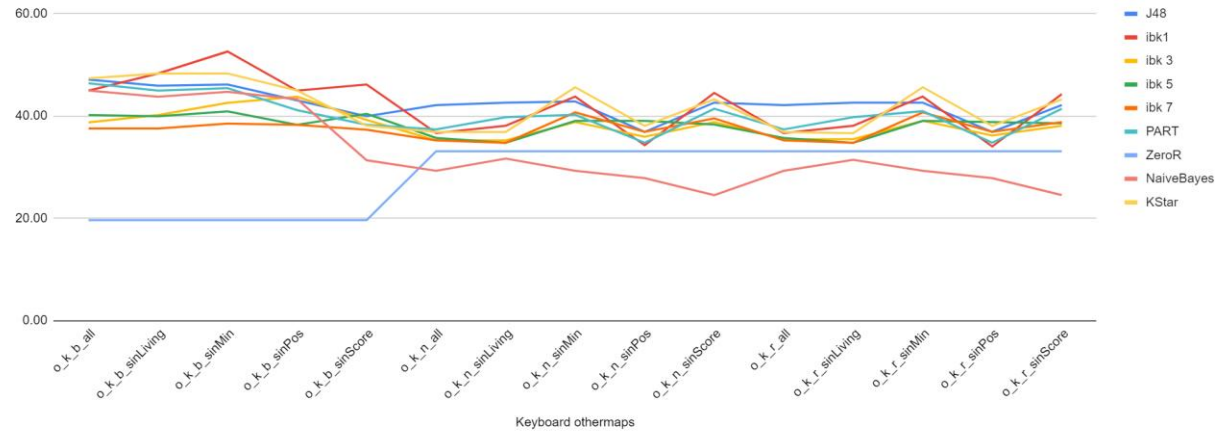


En el caso de los datos obtenido por teclado y evaluado con mapas conocido, los mejores resultados para los distintos filtros se obtienen para el conjunto de datos denominados sinMin, con unas medias de 50.16% para balanceado, 52.2% para normalizado y 52.25% para el caso sin filtrar. Lo que también nos dice que no filtrar los datos suele ser mejor.

Con respecto al algoritmo que mejores resultados ha proporcionado ha sido PART, que de media ha proporcionado un 53.29% de instancias clasificadas con éxito, pero muy de cerca J48 con 53.23%.

Aun así, el pico de éxito al clasificar lo proporciona J48, sin filtrar y con todos los atributos excepto la distancia al fantasma más cercano con un 59.66%.

Keyboard othermaps	J48	IBk 1	IBk 3	IBk 5	IBk 7	PART	ZeroR	NaiveBayes	KStar
o_k_b_all	47.13	44.98	38.76	40.19	37.56	46.41	19.62	44.98	47.38
o_k_b_sinLiving	45.93	48.33	40.19	39.95	37.56	44.98	19.62	43.78	48.33
o_k_b_sinMin	46.17	52.63	42.58	40.91	38.52	45.45	19.62	44.74	48.33
o_k_b_sinPos	43.06	44.98	43.78	38.28	38.28	41.15	19.62	43.30	45.00
o_k_b_sinScore	39.95	46.17	39.23	40.43	37.32	38.28	19.62	31.34	38.10
o_k_n_all	42.14	36.67	35.24	35.71	35.24	37.38	33.10	29.29	36.88
o_k_n_sinLiving	42.62	38.10	35.24	34.76	34.76	39.76	33.10	31.67	36.88
o_k_n_sinMin	42.86	43.81	38.81	39.05	40.71	40.24	33.10	29.29	45.63
o_k_n_sinPos	36.90	34.29	35.95	39.05	36.90	34.76	33.10	27.86	38.06
o_k_n_sinScore	42.62	44.52	38.81	38.33	39.52	41.43	33.10	24.52	43.26
o_k_r_all	42.14	36.67	35.48	35.71	35.24	37.38	33.10	29.29	36.88
o_k_r_sinLiving	42.62	38.10	35.48	34.76	34.76	39.76	33.10	31.43	36.64
o_k_r_sinMin	42.62	43.81	39.05	39.05	40.71	40.95	33.10	29.29	45.63
o_k_r_sinPos	36.90	34.05	36.19	38.81	36.90	34.76	33.10	27.86	38.06
o_k_r_sinScore	42.14	44.29	38.10	38.57	38.81	41.43	33.10	24.52	43.26

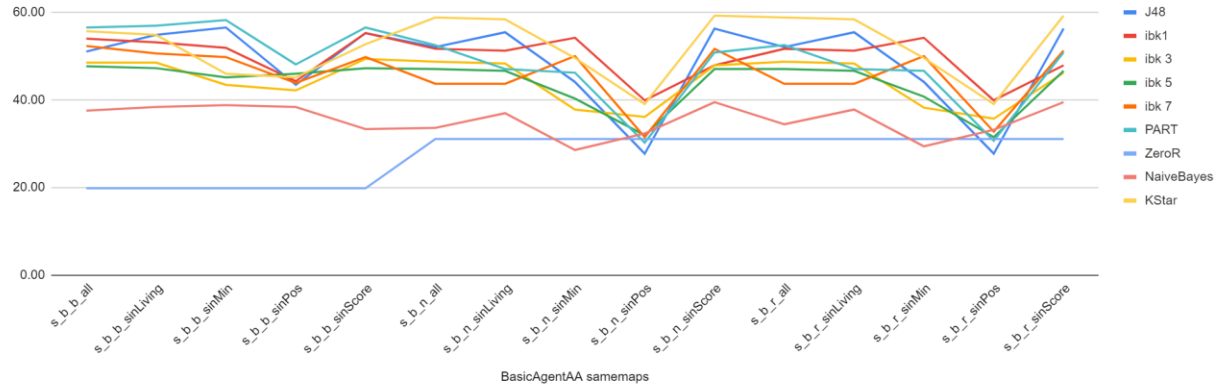


En el caso de los datos obtenidos por teclado y evaluado con mapas nuevos se han obtenido peores resultados con respecto a los mapas conocidos, aunque era de esperar. El conjunto de atributos que mejor ha rendido ha sido otra vez sinMin, con unas medias de 41.33% para balanceado, 38.48% para normalizado y 38.57% para el caso sin filtrar. Aunque en este caso ha dado mejores resultados balanceando las instancias.

Con respecto al algoritmo que mejores resultados ha proporcionado ha sido J48, con un 42.39% de instancias clasificadas con éxito, pero ahora PART se ha quedado atrás con un 40.28%.

Como hemos visto el mejor ha sido para IBk 1, balanceado y sin distancia al fantasma más cercano con un 52.63%.
Lo que podemos concluir es que el mejor set de entrenamiento es sinMin para el agente por teclado.

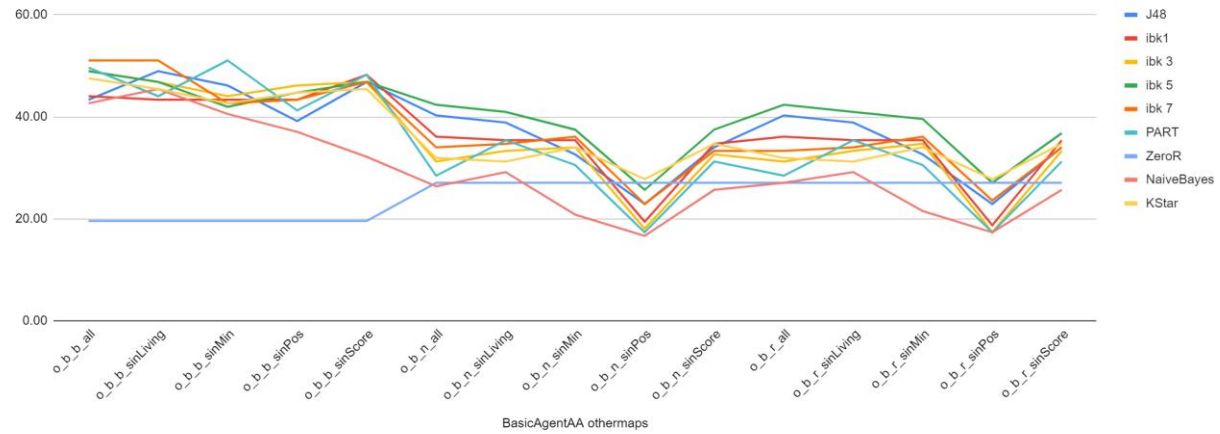
BasicAgentAA samemaps	J48	IBk 1	IBk 3	IBk 5	IBk 7	PART	ZeroR	NaiveBayes	KStar
s_b_b_all	51.05	54.01	48.52	47.68	52.32	56.54	19.83	37.55	55.70
s_b_b_sinLiving	54.85	53.16	48.52	47.26	50.63	56.96	19.83	38.40	54.85
s_b_b_sinMin	56.54	51.90	43.46	45.15	49.79	58.23	19.83	38.82	45.99
s_b_b_sinPos	43.46	44.30	42.19	45.99	43.88	48.10	19.83	38.40	45.15
s_b_b_sinScore	55.27	55.27	49.37	47.26	49.79	56.54	19.83	33.33	52.74
s_b_n_all	52.10	51.68	48.74	47.06	43.70	52.52	31.09	33.61	58.82
s_b_n_sinLiving	55.46	51.26	48.32	46.64	43.70	47.06	31.09	36.97	58.40
s_b_n_sinMin	44.12	54.20	37.82	40.34	50.00	46.22	31.09	28.57	49.58
s_b_n_sinPos	27.73	39.92	36.13	31.93	31.51	30.25	31.09	32.35	39.08
s_b_n_sinScore	56.30	47.90	47.90	47.06	51.68	50.84	31.09	39.50	59.24
s_b_r_all	52.10	51.68	48.74	47.06	43.70	52.52	31.09	34.45	58.82
s_b_r_sinLiving	55.46	51.26	48.32	46.64	43.70	47.06	31.09	37.82	58.40
s_b_r_sinMin	44.12	54.20	38.24	40.76	50.00	46.64	31.09	29.41	49.58
s_b_r_sinPos	27.73	39.92	35.71	31.51	32.77	30.67	31.09	33.19	39.08
s_b_r_sinScore	56.30	47.90	46.22	46.64	51.26	50.84	31.09	39.50	59.24



En el caso de los datos obtenidos por BasicAgentAA y evaluado con mapas conocidos se han obtenido peores resultados con respecto a los jugados por un humano, que era de esperar porque no es un algoritmo perfecto y no es más listo que una persona.

El conjunto de atributos que mejores resultados ha dado ha sido sinScore, con unas medias de 45.83% para balanceado, 46.53% para normalizado y 46.22% para el caso sin filtrar.
Con respecto al algoritmo que mejores resultados ha proporcionado ha sido J48 y IBk1, con un 48.84% y 49.90% de instancias clasificadas con éxito, respectivamente.
A pesar de que en general empeoran el mejor resultado ha sido 59.24% que está muy próximo al mejor de los resultados del agente por teclado con mapas conocidos, este resultado se ha obtenido con KStar, sinScore y para los filtros normalizado y sin filtros.

BasicAgentAA othermaps	J48	IBk 1	IBk 3	IBk 5	IBk 7	PART	ZeroR	NaiveBayes	KStar
o_b_b_all	43.36	44.06	48.95	48.95	51.05	49.65	19.58	42.66	47.55
o_b_b_sinLiving	48.95	43.36	46.85	46.85	51.05	44.06	19.58	45.45	45.45
o_b_b_sinMin	46.15	43.36	44.06	41.96	42.66	51.05	19.58	40.56	42.66
o_b_b_sinPos	39.16	43.36	46.15	44.76	43.36	41.26	19.58	37.06	44.76
o_b_b_sinScore	46.85	48.25	46.85	46.85	46.85	48.25	19.58	32.17	45.45
o_b_n_all	40.28	36.11	31.25	42.36	34.03	28.47	27.08	26.39	31.94
o_b_n_sinLiving	38.89	35.42	33.33	40.97	34.72	35.42	27.08	29.17	31.25
o_b_n_sinMin	32.64	35.42	34.03	37.50	36.11	30.56	27.08	20.83	34.03
o_b_n_sinPos	22.92	19.44	18.06	25.69	22.92	17.36	27.08	16.67	27.78
o_b_n_sinScore	34.03	34.72	32.64	37.50	33.33	31.25	27.08	25.69	34.72
o_b_r_all	40.28	36.11	31.25	42.36	33.33	28.47	27.08	27.08	31.94
o_b_r_sinLiving	38.89	35.42	33.33	40.97	34.03	35.42	27.08	29.17	31.25
o_b_r_sinMin	32.64	35.42	34.72	39.58	36.11	30.56	27.08	21.53	34.03
o_b_r_sinPos	22.92	18.75	17.36	27.08	23.61	17.36	27.08	17.36	27.78
o_b_r_sinScore	34.03	35.42	33.33	36.81	34.03	31.25	27.08	25.69	34.72



En el caso de los datos obtenidos por el BasicAgentAA y evaluado con mapas desconocidos, se han obtenido resultados más bajos que con los conocidos, pero la diferencia es menor que la que aparecía con el agente por teclado. Con respecto al conjunto de atributos que mejor han clasificado ha sido sinLiving, con unas medias de 43.27% para balanceado, 34.38% para normalizado y 34.29% para el caso sin filtrar. El filtro que mejores resultados ha proporcionado han sido los balanceados y el algoritmo que mejores resultados ha proporcionado ha sido IBk5, con un 40.01% de instancias clasificadas con éxito.
Los mejores resultados han sido para IBk 7, balanceado, con todos los atributos y también sinLiving, otro que ha proporcionado los mismos resultados ha sido PART, balanceado y sinMin, estas tres combinaciones han dado un 51.05% de éxito

Para el agente que juega solo del Tutorial 1 los mejores resultados son con los atributos sinMin, sinScore o sinLiving, normalizados o balanceados y con el algoritmo KStar, IBk 7 o PART. Por lo general lo mejor es balancear las instancias, y el algoritmo dependerá del caso en el que nos encontremos, pero consideramos que la mejor combinación será KStar, normalizado y sinScore.

En resumen, las mejores combinaciones han sido:

	Atributos	**Filtro**	**Algoritmo**	**Éxito**
Keyboard	sinMin	Sin filtrar	J48	59.66%
BasicAgentAA	sinScore	Normalizado	KStar	59.24%

Además, se puede ver que se produce poco sobreajuste, comparando los samemaps con los othermaps.

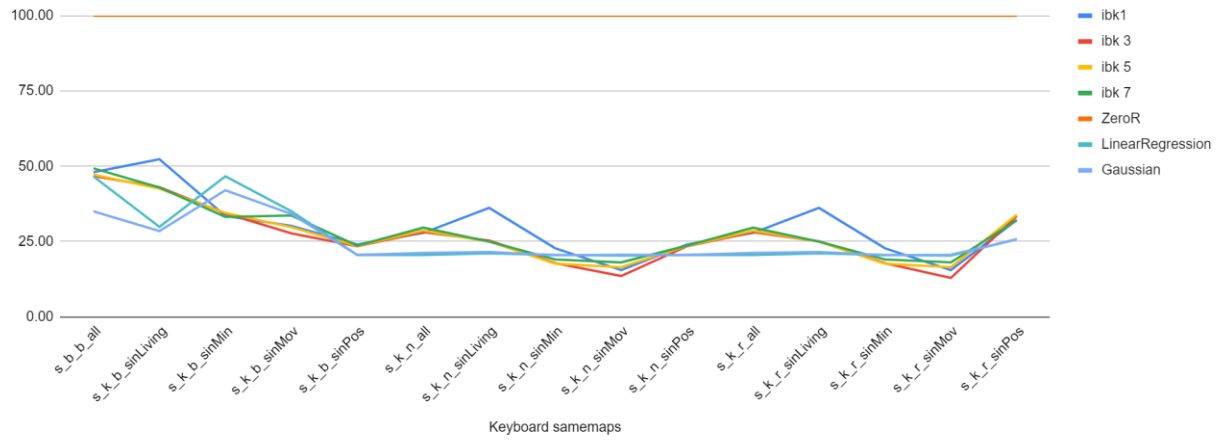
Fase 3: Predicción

En esta fase se ha seguido un procedimiento parecido al de la fase anterior. En este caso se tiene en cuenta el '*root relative squared error*' que, en este caso, cuanto menor sea mejor será la predicción del modelo. Los algoritmos usados han sido los siguientes:

- **IBK 1**: Es un algoritmo bastante sencillo y conocido, y en algunos casos puede dar resultados bastante buenos, en este caso hemos usado k =1
- **IBK 3**: Utilizar k=1 puede ser algo básico, por lo que hemos decidido que aumentar k sería un paso bastante lógico.
- **IBK 5**: Debido al alto número de instancias tener un k alto puede dar buenos resultados por lo que k=5 podría generar datos relevantes.
- **IBK 7**: Para terminar con este algoritmo hemos decidido que sería interesante utilizar k=7 ya que hemos visto que aumentar la k ha dado buenos resultados incrementalmente, por lo que hemos introducido esta variación del algoritmo como último caso.
- **ZeroR**: Es un algoritmo muy simple pero conocido, al menos se podrá utilizar para comparar y entender mejor cómo se comportan los algoritmos.
- **LinearRegression**: Haciendo diversas pruebas hemos visto que este algoritmo da muy buenos resultados, y sobre todo estables, que es algo que obviamente buscamos, ya que podemos controlar mejor cómo se comportará si lo usamos para los demás mapas.
- **Gaussian**: Este último algoritmo ha sido elegido ya que también es bastante estable y los resultados son aceptables, cosa que nos será de gran utilidad en siguientes pasos.

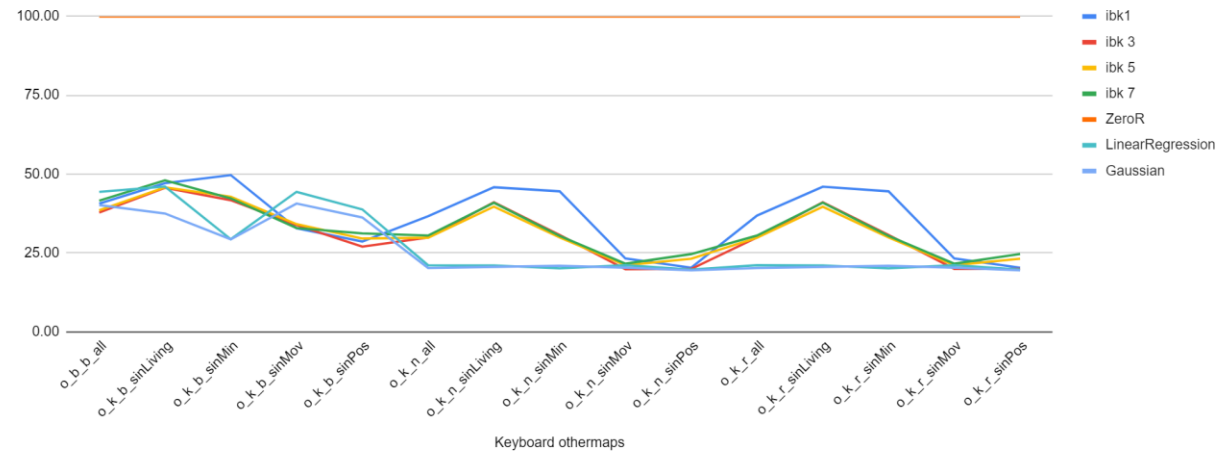
El método de nombrado de los ficheros es idéntico al de la fase anterior por lo que no se explicará aquí.

Keyboard samemaps	IBk 1	IBk 3	IBk 5	IBk 7	ZeroR	LinearRegression	Gaussian
s_b_b_all	48.03	46.73	47.20	49.32	100.00	46.63	35.03
s_k_b_sinLiving	52.35	43.06	42.59	42.97	100.00	29.86	28.49
s_k_b_sinMin	33.51	34.37	34.50	33.17	100.00	46.67	42.03
s_k_b_sinMov	30.20	27.74	29.75	33.66	100.00	35.07	34.17
s_k_b_sinPos	24.02	23.43	23.54	23.67	100.00	20.54	20.54
s_k_n_all	27.86	28.12	28.64	29.62	100.00	20.55	21.22
s_k_n_sinLiving	36.18	25.36	25.01	25.00	100.00	21.09	21.52
s_k_n_sinMin	22.78	17.85	17.66	19.05	100.00	20.54	20.47
s_k_n_sinMov	15.53	13.57	16.46	18.07	100.00	20.34	20.64
s_k_n_sinPos	24.02	23.42	23.54	23.67	100.00	20.54	20.54
s_k_r_all	27.86	28.12	28.60	29.62	100.00	20.55	21.22
s_k_r_sinLiving	36.18	25.07	25.00	25.00	100.00	21.09	21.52
s_k_r_sinMin	22.78	17.85	17.66	19.05	100.00	20.54	20.47
s_k_r_sinMov	15.53	12.90	16.48	18.10	100.00	20.34	20.64
s_k_r_sinPos	32.19	33.46	33.88	32.10	100.00	25.80	25.71



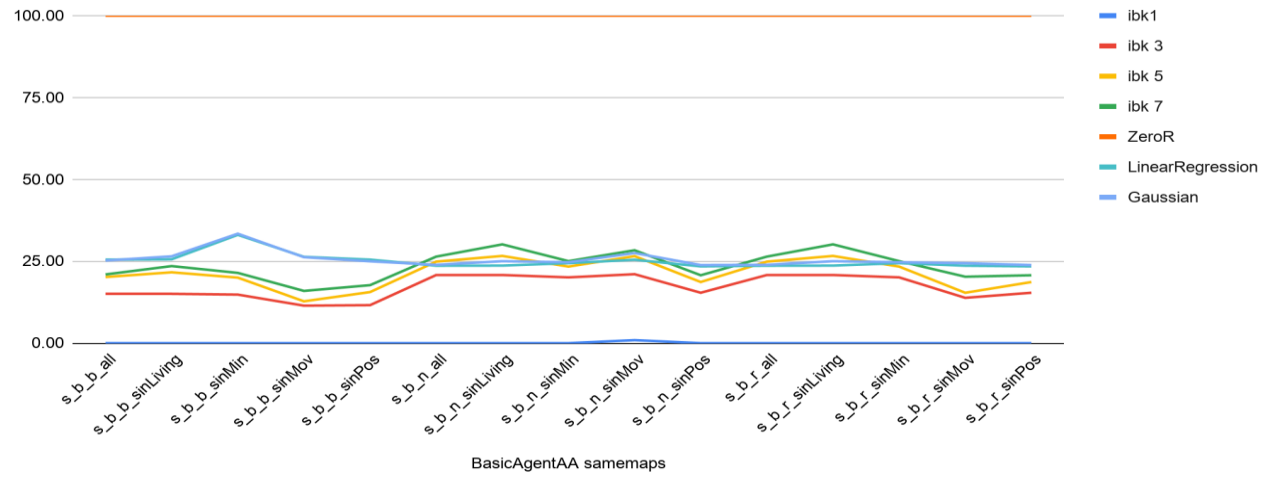
Para la predicción del agente de teclado con samemaps tenemos que el mejor algoritmo está bastante disputado entre LinearRegression y Gaussian, ya que los resultados son casi idénticos y varían en una unidad en los casos más extremos, por lo que, a nuestro criterio la elección de algoritmo en este caso puede ser de cualquiera de los dos, aun así, con unos datos más extensos se podría llegar a una mejor conclusión. El mejor valor se da en IBK-3, en este caso de un 12.9% con el fichero sin filtros y sin el atributo de movimiento de Pac-man. En este caso IBK-3 da un mínimo bastante bueno así que podría ser un competidor con los dos algoritmos anteriores.

Keyboard othermaps	IBk 1	IBk 3	IBk 5	IBk 7	ZeroR	LinearRegression	Gaussian
o_b_b_all	40.61	37.75	38.46	41.54	100.00	44.34	40.13
o_k_b_sinLiving	47.08	45.63	45.79	47.98	100.00	46.01	37.46
o_k_b_sinMin	49.66	41.67	42.77	42.25	100.00	29.30	29.27
o_k_b_sinMov	32.79	33.79	34.09	32.85	100.00	44.34	40.63
o_k_b_sinPos	28.56	26.93	29.52	31.14	100.00	38.76	36.20
o_k_n_all	36.61	29.85	29.77	30.48	100.00	21.05	20.15
o_k_n_sinLiving	45.81	41.04	39.63	40.87	100.00	20.98	20.52
o_k_n_sinMin	44.50	30.66	29.89	30.35	100.00	20.04	20.87
o_k_n_sinMov	23.18	19.75	21.02	21.53	100.00	21.05	20.28
o_k_n_sinPos	20.27	19.85	23.11	24.59	100.00	19.67	19.42
o_k_r_all	36.80	29.85	29.77	30.48	100.00	21.05	20.15
o_k_r_sinLiving	45.96	41.04	39.62	40.89	100.00	20.98	20.52
o_k_r_sinMin	44.50	30.64	29.89	30.36	100.00	20.04	20.87
o_k_r_sinMov	23.18	19.88	21.06	21.53	100.00	21.05	20.28
o_k_r_sinPos	20.27	19.85	23.12	24.62	100.00	19.67	19.42



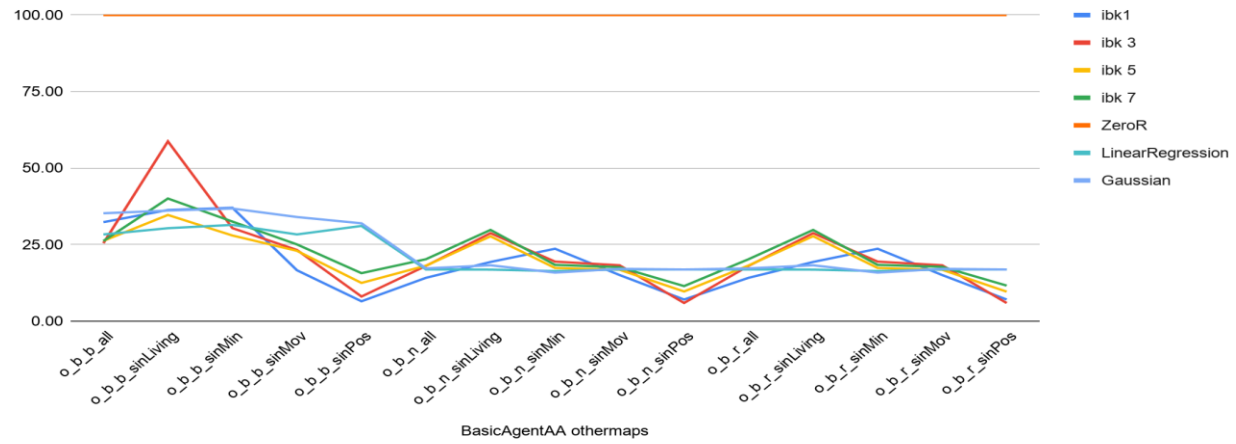
Esta vez se estudia othermaps con el agente por teclado. Los resultados más representativos se dan en Gaussian, y también su mejor dato, con un 19.42%, aunque LinearRegression e IBK-3 dan muy buenos resultados en algunos casos. Cabe destacar que los datos son algo mejores que con samemaps, no son mejoras muy representativas, pero se tendrán en cuenta.

BasicAgentAA samemaps	IBk 1	IBk 3	IBk 5	IBk 7	ZeroR	LinearRegression	Gaussian
s_b_b_all	0.00	15.06	20.19	20.98	100.00	25.51	25.22
s_b_b_sinLiving	0.00	15.06	21.63	23.53	100.00	25.60	26.52
s_b_b_sinMin	0.00	14.82	20.00	21.45	100.00	33.07	33.43
s_b_b_sinMov	0.00	11.45	12.77	15.95	100.00	26.36	26.25
s_b_b_sinPos	0.00	11.60	15.59	17.71	100.00	25.51	24.98
s_b_n_all	0.00	20.80	24.90	26.42	100.00	23.64	23.93
s_b_n_sinLiving	0.00	20.80	26.64	30.16	100.00	23.68	25.02
s_b_n_sinMin	0.00	20.08	23.38	25.05	100.00	24.43	24.72
s_b_n_sinMov	0.91	21.06	26.56	28.37	100.00	25.47	27.51
s_b_n_sinPos	0.00	15.39	18.67	20.73	100.00	23.44	23.85
s_b_r_all	0.00	20.80	24.89	26.42	100.00	23.64	23.93
s_b_r_sinLiving	0.00	20.80	26.63	30.16	100.00	23.68	25.02
s_b_r_sinMin	0.00	20.08	23.38	25.05	100.00	24.43	24.72
s_b_r_sinMov	0.00	13.84	15.38	20.29	100.00	23.64	24.41
s_b_r_sinPos	0.00	15.39	18.66	20.73	100.00	23.44	23.85



Estos datos nos han sorprendido bastante ya que IBK-1 aporte un valor de 0% en todos los casos, que es un dato perfecto. Obviamente esto no se debe tener en cuenta para elegir un fichero y algoritmo de predicción. Tenemos la hipótesis de que ha ocurrido un overfitting con estos mapas. También se puede observar que los datos son bastante buenos para ser un agente 'inteligente' con un comportamiento muy simple y que puede originar numerosos errores o situaciones sin salida, por lo que la teoría se soporta bastante.

BasicAgentAA othermaps	IBk 1	IBk 3	IBk 5	IBk 7	ZeroR	LinearRegression	Gaussian
o_b_b_all	32.33	25.46	26.21	26.18	100.00	28.32	35.24
o_b_b_sinLiving	36.31	58.73	34.68	40.06	100.00	30.36	36.10
o_b_b_sinMin	37.03	30.38	27.97	32.52	100.00	31.43	36.78
o_b_b_sinMov	16.61	23.24	22.96	25.05	100.00	28.32	34.01
o_b_b_sinPos	6.47	8.03	12.47	15.66	100.00	31.10	31.96
o_b_n_all	14.15	18.04	18.12	20.23	100.00	16.87	17.22
o_b_n_sinLiving	19.32	28.74	27.70	29.80	100.00	16.82	18.24
o_b_n_sinMin	23.64	19.44	17.36	18.36	100.00	16.34	15.84
o_b_n_sinMov	15.05	18.24	16.77	17.62	100.00	16.87	17.08
o_b_n_sinPos	7.05	5.93	9.68	11.44	100.00	16.87	16.86
o_b_r_all	14.15	18.04	18.18	20.23	100.00	16.87	17.22
o_b_r_sinLiving	19.32	28.74	27.74	29.79	100.00	16.82	18.24
o_b_r_sinMin	23.64	19.44	17.36	18.36	100.00	16.34	15.84
o_b_r_sinMov	15.05	18.24	16.77	17.73	100.00	16.87	17.08
o_b_r_sinPos	7.05	5.90	9.61	11.60	100.00	16.87	16.86



Por último, el agente inteligente con othermaps también ha dado mejores resultados de los esperados. El mínimo lo vuelve a ostentar IBK-3 con un 5.9% que es un dato muy bueno comparado a los anteriores datos, el fichero que ha generado este valor ha sido el balanceado sin atributo de posición, podemos empezar a ver un patrón en cuanto a filtros y atributos que benefician al resultado. En definitiva, volvemos a pensar que la elección de mapas ha sido bastante 'ideal' para este algoritmo y que a la hora de extrapolarlo a otros mapas el resultado no sería tan bueno como estamos observando en estas gráficas.

Fase 4: Construcción de un Agente Automático

Como se ha concluido en la Fase 2 las mejores combinaciones de algoritmo, filtro y conjunto de atributos para predecir la siguiente acción de Pac-man con un relativo buen éxito son:

> El algoritmo de árboles de decisión J48 con las coordenadas de Pac-man, el número de fantasmas vivos y la puntuación de ese estado como atributos, sin realizar ninguna modificación sobre sus valores (sin aplicar filtros).
>
> Y el algoritmo de aprendizaje vago KStar, con las coordenadas de Pac-man, el número de fantasmas vivos y la distancia al más cercano como atributos, todos estos con sus valores normalizados.

Se ha generado un modelo para ambas combinaciones con los nombres, j48.model y kstar.model, respectivamente. Ambos modelos no son capaces de ganar la partida de una manera eficiente, sobre todo el de KStar cuyo único movimiento es desplazarse hacia el este y a veces recorre el borde, pero no alcanza a los fantasmas.

Sin embargo, el de J48 a pesar de que inicialmente siempre se desplaza hacia el sur y después al este hasta dar con un muro, eventualmente es capaz de abandonar su posición junto a la pared para perseguir a algún fantasma de manera breve, a veces lo alcanza y otras pierde el interés hasta darse contra otra barrera.

Por tanto, el mejor modelo que hemos conseguido generar ha sido el j48.model, aunque está lejos de ser un modelo óptimo para la tarea de comerse todos los fantasmas.

Nuestro mejor modelo por lo tanto no es mejor que una persona utilizando el agente por teclado, dado que una persona por muy poca experiencia en el juego que tenga es capaz de ver las rutas óptimas de manera intuitiva, cosa que el modelo no es capaz de calcular.

Con respecto al BasicAgentAA implementado en el Tutorial 1, en la mayoría de los casos nuestro modelo no encontraría la solución antes que este, pero en algunos casos el agente BasicAgentAA se queda completamente bloqueado y no ganará al modelo.

Preguntas

1. **¿Qué diferencias hay a la hora de aprender esos modelos con instancias provenientes de un agente controlado por un humano y uno automático?**
   El agente humano tendrá un patrón menos predecible, aunque (normalmente) más eficiente. Por otro lado, el agente automático depende mucho de la calidad con la que se ha creado, es decir, podemos hacer un agente que siempre vaya a derechas; si se usa como entrenamiento, será muy predecible, pero agente resultante será muy malo.
   Si usamos un agente automático que funcione medianamente bien, podemos obtener mejores resultados que con un agente controlado por un humano, aunque el tiempo invertido en crear un buen agente automático puede ser muy grande y que no merezca la pena realizar aprendizaje automático porque sea lo suficientemente eficaz.

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?**
   Si se quisiera transformar un problema de regresión a clasificación podríamos discretizar los valores de los atributos, es decir, dividir los valores numéricos en diferentes rangos, creando intervalos que se puedan utilizar como valores discretos, de esta manera cuando elija una clase en la clasificación tendremos un rango de valores del que se escogerá según un determinado criterio el valor de la regresión.
   Si se perfeccionara suficiente la predicción de la puntuación, nos podría indicar cuando nos acercamos al final o incluso llevar por el camino óptimo comiéndose los fantasmas y puntos del tablero.
3. **¿Qué ventajas puede aportar predecir la puntuación respecto a la clasificación de la acción? Justifica tu respuesta.**
   Si nuestro objetivo es priorizar el aumentar la puntuación, si se introduce por ejemplo la acción de comer comida, puede ser interesante predecir la puntuación sobre las acciones de Pac-man. Aunque por ahora, nuestro objetivo principal es comer todos los fantasmas que nos encontremos en el tablero por lo que aumentar la puntuación no sería tan efectivo como en el caso anterior.
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?**
   Puede ser que mejore la clasificación ya que se puede tener una idea general del estado del juego y como la acción de comerse un fantasma aumenta la puntuación considerablemente. Esto puede hacer que se priorice aún más el comer un fantasma, y que la clasificación sea más efectiva.

Conclusiones

En definitiva, hemos encontrado esta práctica bastante completa y complicada. Encontrar una combinación de atributos, filtros y algoritmos que den un buen resultado y que funcionen bien con las instancias de prueba y entrenamiento es una tarea más compleja de lo que pensábamos, aunque nos ha acercado un poco más a conocer cómo funciona el aprendizaje autónomo. Por otro lado, también nos han surgido nuevas preguntas, como si se podría implementar este problema con una red neuronal, o cómo puede funcionar con aprendizaje mediante refuerzo. Y aunque no hayamos podido desarrollar un agente que juegue de manera perfecta a Pac-man, hemos obtenido nuestra primera aproximación, que ha hecho que nos interesemos en optimizar y mejorar este problema con nuevas técnicas.