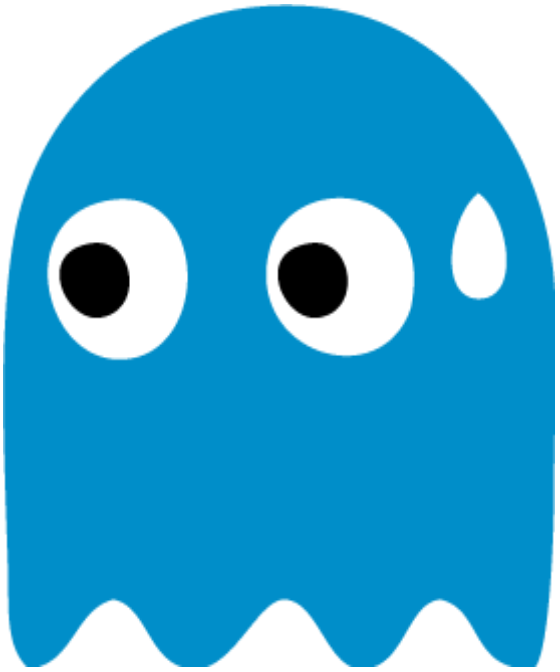
Aprendizaje Automático

Práctica 1



uc3m | Universidad Carlos III de Madrid

Ricardo Prieto Álvarez 100386267

ÍNDICE

INTRODUCCIÓN

Este documento contiene la descripción del procedimiento de elaboración de las distintas fases para la aplicación de técnicas variadas de aprendizaje automático enfocado al juego Pac-Man.

Estos modelos generados mediante la herramienta Weka serán utilizados para construir un agente capaz de jugar de manera automática.

La fase 1 de esta práctica describe la función utilizada para capturar las instancias del juego, siendo este jugado por teclado y por la IA implementada en el tutorial 1, además de la técnica utilizada para la captura de datos pertenecientes al tick siguiente.

En cuanto a la fase 2, esta se enfoca en el preprocesado, la experimentación y la elección del modelo final basándose en los conjuntos de entrenamiento y de test.

La fase 3 sigue un procedimiento similar al de la fase 2, salvo que el enfoque está dirigido a la predicción de los valores futuros de la puntuación basándose en modelos de regresión de aprendizaje automático supervisado.

En la última fase se justifica la elección del modelo finalista y se realiza una comparación de este mismo modelo entrenado con los dos conjuntos de entrenamiento distintos.

Por último se responde a las 4 preguntas propuestas y, finalmente, se elabora una conclusión resumiendo todo lo apreciado en esta práctica.

Fase 1

El método utilizado para la extracción de características del estado del juego tiene el nombre de printLineData. Es decir, este método ha sido reutilizado de las prácticas anteriores.

El método almacena 24 atributos por instancia, escribe las cabeceras necesarias para que sea un archivo .arff correcto. Si el fichero de captura de datos no está creado, este lo crea escribiendo dichas cabeceras. En este punto, cada iteración proporciona un valor para cada atributo, que es implementado en una lista de listas con el objetivo de hacer uso de la función savetxt() de la librería numpy.

Los 24 atributos seleccionados son:

- **pacmanDirec:** La dirección actual del pacman
- **pacmanXpos:** Coordenada x actual del pacman
- **pacmanYpos:** Coordenada y actual del pacman
- **LivingGhost1:** True/False si el fantasma 1 está o no vivo
- **LivingGhost2:** True/False si el fantasma 2 está o no vivo
- **LivingGhost3:** True/False si el fantasma 3 está o no vivo
- **LivingGhost4:** True/False si el fantasma 4 está o no vivo
- **ghost1Xpos:** Coordenada x actual del fantasma 1
- **ghost1Ypos:** Coordenada y actual del fantasma 1
- **ghost2Xpos:** Coordenada x actual del fantasma 2
- **ghost2Ypos:** Coordenada y actual del fantasma 2
- **ghost3Xpos:** Coordenada x actual del fantasma 3
- **ghost3Ypos:** Coordenada y actual del fantasma 3
- **ghost4Xpos:** Coordenada x actual del fantasma 4
- **ghost4Ypos:** Coordenada y actual del fantasma 4
- **ghost1Dist:** Distancia euclídea entre el fantasma 1 y el pacman
- **ghost2Dist:** Distancia euclídea entre el fantasma 2 y el pacman
- **ghost3Dist:** Distancia euclídea entre el fantasma 3 y el pacman
- **ghost4Dist:** Distancia euclídea entre el fantasma 4 y el pacman
- **currentScore:** La puntuación del tick actual
- **pacmanNextXPosition:** Coordenada x del pacman del siguiente tick
- **pacmanNextYPosition:** Coordenada y del pacman del siguiente tick
- **nextScore:** La puntuación del tick siguiente (El valor a predecir)
- **action:** La acción que decidirá tomar el pacman (La clase a clasificar)

Para poder determinar la puntuación del siguiente score se toma el parámetro nextState que se proporciona al método printLineData() y se llama al método getScore(), esto devuelve el valor del score del estado del tick siguiente. Para poder obtener el estado del tick siguiente es necesario hacer una llamada al método generateSuccessor(agentIndex, action) para cuando el agentIndex sea equivalente a 0, indicando así que se trata del agente pacman. Para finalmente almacenar el estado futuro al igual que se hace con gameState, se llama a deepCopy(). Finalmente, se ejecuta printLineData(currentState, nextState) donde currentState lleva el estado del tick actual y nextState el estado del tick siguiente, como ya se ha explicado.

Los mapas utilizados para la captura de las instancias son los siguientes:

Training maps y test maps	oneHunt new (personalizado) testClassic openClassic originalClassic
Test other maps	bugHunt newmap openHunt smallHunt trickyClassic

Fase 2

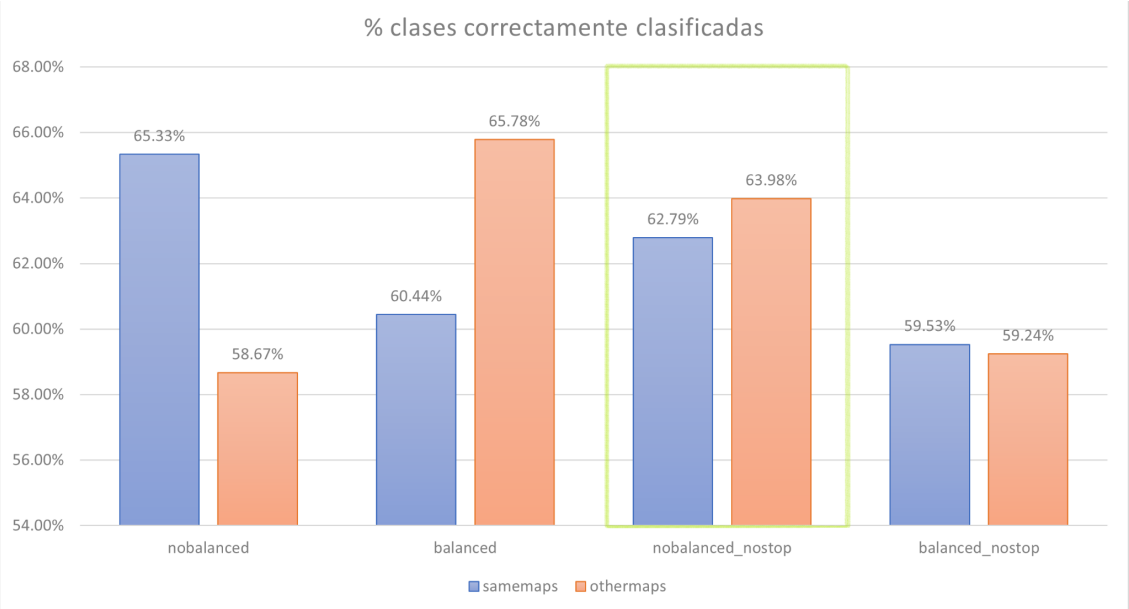
En esta fase se irán mostrando las partes más características del proceso de experimentación que ha sido llevado a cabo para finalmente escoger el modelo más óptimo.

Para plantear la experimentación se ha hecho uso del conjunto de datos captados por teclado, manteniendo como suposición que el dataset basado en el tutorial 1 mostrará los mismos resultados o ligeramente mejores.

Para la selección de atributos determinantes para el modelo de clasificación se tendrán en cuenta todos ellos en un principio y se procederá a realizar una serie de experimentos combinando balanceado y limpieza del dataset (árboles de decisión no necesitan normalización) usando el algoritmo RandomForest, de tal manera que se pueda conocer cuál es la característica que deben presentar los atributos para mostrar mejores resultados en las métricas de evaluación.

- nobalanced ≡ clases no balanceadas
- balanced ≡ clases balanceadas
- nostop ≡ data set limpio de instancias con valores Stop en los atributos action y pacmanDirec

RandomForest	**test: samemaps**	**test: othermaps**
	% Clases correctamente clasificadas	% Clases correctamente clasificadas
nobalanced	65.33	58.67
balanced	60.44%	65.78%
nobalanced_nostop	62.79%	63.98%
balanced_nostop	59.53%	59.24%



Dada la tabla anterior, se escoge la mejor combinación de características de los datos probadas (marcada en verde) y se realizan 6 pruebas, cada una de ellas corresponde a una variación de atributos, es decir, la primera variación contendrá todos los atributo y las demás serán el mismo conjunto pero descartando algunos atributos determinados. Las variaciones son las siguientes:

- **Variación 1 (todo):** pacmanDirec, pacmanXpos, pacmanYpos, LivingGhost1, LivingGhost2, LivingGhost3, LivingGhost4, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, ghostDist1, ghostDist2, ghostDist3, ghostDist4, action.

- **Variación 2 (todo - ghostDist - pacmanDirec):** pacmanXpos, pacmanYpos, LivingGhost1, LivingGhost2, LivingGhost3, LivingGhost4, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, action.

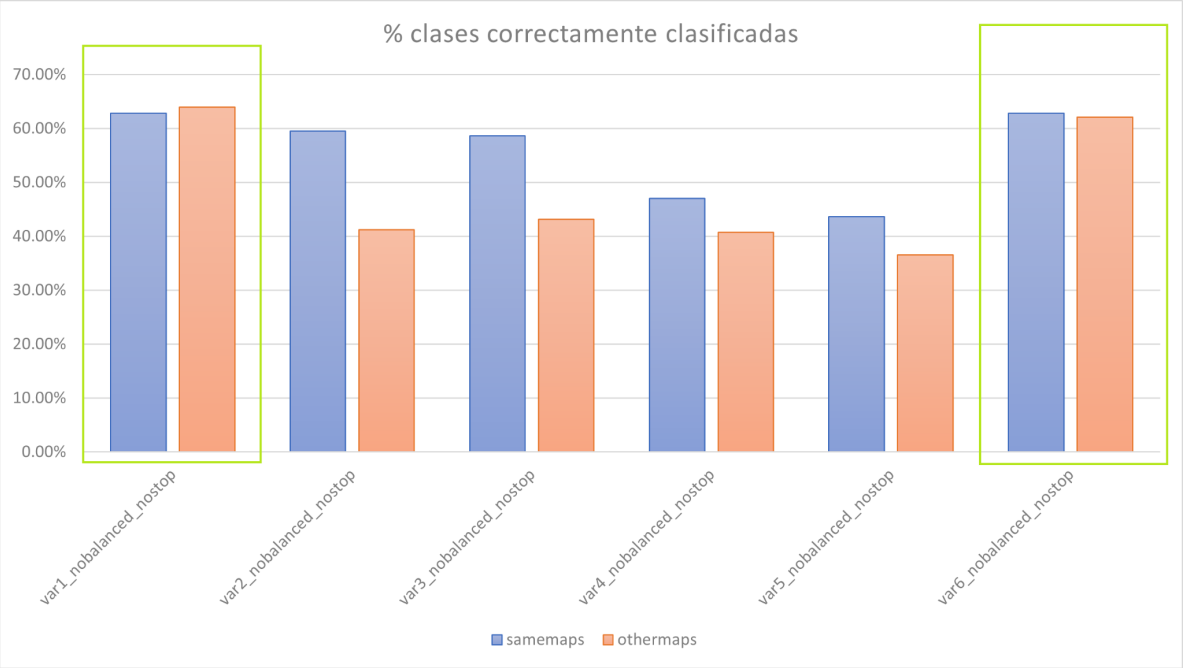
- **Variación 3 (solo posiciones de pacman y ghosts):** pacmanXpos, pacmanYpos, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, action.

- **Variación 4 (solo posiciones ghosts):** ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, action.

- **Variación 5 (solo ghostDist):** ghostDist1, ghostDist2, ghostDist3, ghostDist4, action.
- **Variación 6 (Todo - livingGhosts):** pacmanDirec, pacmanXpos, pacmanYpos, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, ghostDist1, ghostDist2, ghostDist3, ghostDist4, action.

Tomando los datos sin balancear pero sí limpiados, se harán tantas pruebas como variaciones propuestas sobre el algoritmo RandomForest, de tal manera que se pueda seleccionar la variación de atributos con la que realizar la experimentación sobre los demás algoritmos de clasificación. El nombre de la prueba es var<i>_nobalanced_nostop, siendo <i> el número de identificación de la variación a probar.

RandomForest	**test: samemaps**	**test: othermaps**
	% Clases correctamente clasificadas	% Clases correctamente clasificadas
var1_nobalanced_nostop	62.79%	63.98%
var2_nobalanced_nostop	59.53%	41.23%
var3_nobalanced_nostop	58.60%	43.13%
var4_nobalanced_nostop	46.98%	40.76%
var5_nobalanced_nostop	43.63%	36.57%
var6_nobalanced_nostop	62.79%	62.1%



Se pueden observar, marcados en verde en la tabla superior, cuáles son los conjuntos de atributos que mejores resultados ofrece para el algoritmo de RandomForest. A partir de aquí, se tomará la variación 1[1] para hacer la evaluación de los demás algoritmos de clasificación. Las evaluaciones tendrán que realizarse para atributos normalizados y no normalizados ya que algunos de estos modelos no basados en árboles de decisión exigen datos de entrada normalizados.

[1] Durante las pruebas realizadas con el agente automático y wekaI, resultó más sencillo eliminar los atributos Living Ghosts, ya que mostraba un error en las variables con valores True/False. Por lo tanto, en las tablas de evaluación que se muestran a continuación, aparecerán dos filas para la variación 6 de atributos.

En la tabla inferior se pueden observar marcados en negrita los porcentajes de clases correctamente clasificadas ≥ 60%. En el mismo color son los valores dados por el mismo modelo entrenado con el mismo conjunto de datos pero evaluado con los dos tipos de conjuntos de test existentes.

Test: samemaps

	Simple Logistic	**1-NN**	**2-NN**	**3-NN**	**4-NN**	**NaiveBayes**	**SMO**	**J48**	**RandomForest**	**PART**	**REPTree**
var1_norm_balanced	53.33	53.33	52	48.89	49.78	45.33	57.33	46.67	53.33	56	49.33
var1_norm_nobalanced	47.11	36.89	38.67	38.22	41.78	39.11	54.22	53.78	54.67	50.22	56.89
var1_norm_balanced_nostop	52.56	52.56	52.1	52.56	51.63	43.72	58.14	46.51	52.56	46.51	47.91
var1_norm_nobalanced_nostop	56.74	57.67	56.74	**60**	59.53	47.91	57.21	55.35	57.21	49.3	55.81
var1_nonorm_nobalanced	57.33	58.22	56.44	59.11	54.67	44	56.44	**65.33**	**65.33**	**64**	**60.44**
var1_nonorm_balanced	57.78	53.33	51.56	50.22	49.33	46.22	56.89	50.22	**60.44**	55.11	48.44
var1_nonorm_nobalanced_nostop	58.14	57.21	56.74	**60.47**	54.42	48.84	56.74	**65.58**	**62.79**	**61.4**	56.74
var1_nonorm_balanced_nostop	57.21	56.74	54.42	55.35	54.88	47.91	57.67	56.28	59.53	46.98	55.81
var6_nonorm_balanced	56	55.11	50.22	50.22	50.67	50.67	56	52	**56.89**	46.78	48.44
var6_nonorm_nobalanced_nostop	56.28	57.21	56.74	59.1	56.28	49.3	56.28	59.53	**62.79**	58.6	56.74

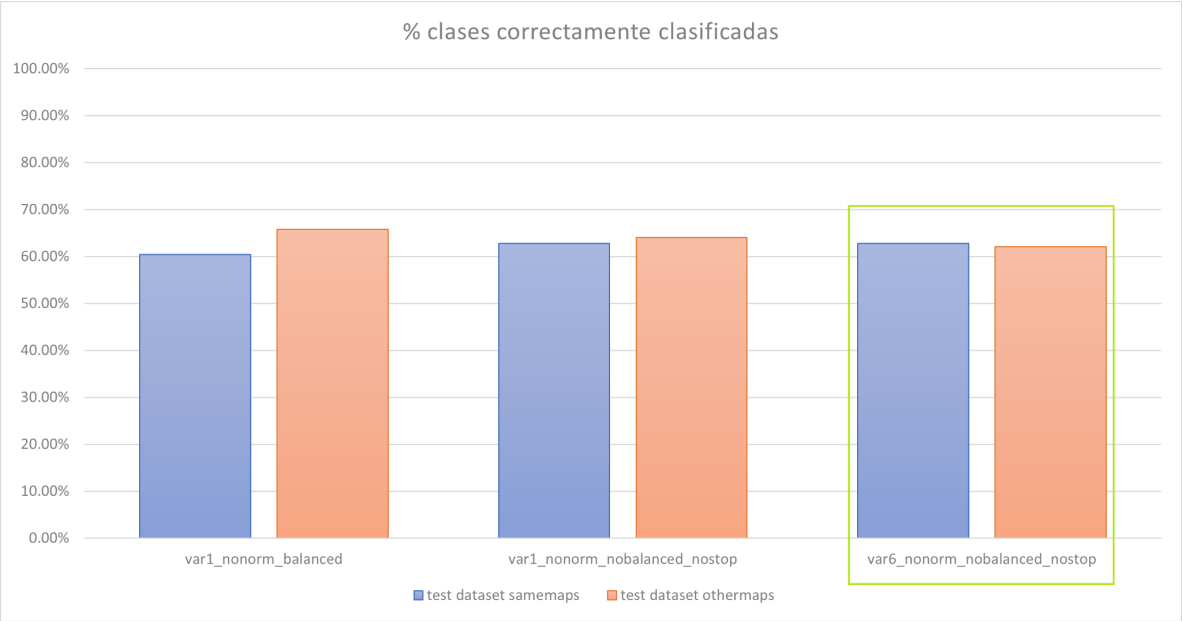
Test: othermaps

	Simple Logistic	1-NN	2-NN	3-NN	4-NN	NaiveBayes	SMO	J48	RandomForest	PART	REPTree
var1_norm_balanced	53.78	46.67	54.67	45.33	44.44	45.33	**62.22**	52.89	**61.33**	44.44	59.56
var1_norm_nobalanced	59.11	40.44	46.22	51.56	54.22	43.11	**64.89**	41.78	**63.56**	31.11	**62.22**
var1_norm_balanced_nostop	56.4	33.18	32.23	38.87	43.13	51.18	**64.93**	38.86	54.51	40.28	41.71
var1_norm_nobalanced_nostop	**63.03**	43.60	47.88	54.03	55.92	45.5	**67.3**	41.23	**65.88**	43.61	**65.4**
var1_nonorm_nobalanced	**64.44**	50.67	48.44	52.89	56.89	58.67	**64.89**	40.89	58.67	44.89	**62.22**
var1_nonorm_balanced	48	49.78	54.67	48	46.22	55.56	**61.78**	50.67	65.78	46.22	56.44
var1_nonorm_nobalanced_nostop	**63.98**	50.72	53.08	54.50	58.77	59.72	**65.88**	41.23	**63.98**	48.34	**64.46**
var1_nonorm_balanced_nostop	**60.19**	41.23	36.2	41.71	43.13	**61.61**	**65.4**	39.34	59.24	38.29	48.82
var6_nonorm_balanced	**64.44**	52.44	55.56	47.56	45.33	54.67	**64**	55.11	**64**	47.11	56.44
var6_nonorm_nobalanced_nostop	**67.78**	49.76	55.92	**62.1**	**65.4**	**63.03**	**65.4**	47.39	**62.1**	50.71	**64.46**

Después de haber realizado todo tipo de experimentaciones, se determinan unos modelos que parecen ser los más generalizables según los datasets de entrenamiento y de test, los cuales se encuentran marcados en verdes en las tablas mostradas más arriba. Estos son elegidos debido a que presentan un porcentaje de clases correctamente clasificadas superior al 60% (ya que ningún modelo ha conseguido obtener un valor superior al 70%) para ambos tipos de conjuntos de test. Es decir, se escogerán los modelos que presenten un valor ≥ 60% en el conjunto de test *samemaps* y *othermaps*.

De estos 2 modelos se debe escoger uno, para ello se hallará la diferencia del % de clases correctamente clasificadas que ofrece cada uno de ellos según el conjunto de test usando los mismos mapas que en el conjunto de entrenamiento y el conjunto de test que usa otros distintos.

RandomForest	test dataset samemaps	test dataset othermaps	|samemaps - othermaps|
var1_nonorm_balanced	60.44%	65.78%	5.34%
var1_nonorm_nobalanced_nostop	62.79%	63.98%	1.19%
var6_nonorm_nobalanced_nostop	62.79%	62.1%	0.69%



Como se observa, el conjunto de entrenamiento entrenado por RandomForest que presenta la variación 6 de atributos, no siendo estos normalizados ni balanceados, pero sí limpiados de instancias ruido de Stop, presenta la menor variación de porcentaje de clases correctamente clasificadas.

A partir de aquí, se corrobora esta elección para los conjuntos de instancias tomadas por la IA programada en el tutorial 01 que tienen las mismas características que el conjunto de instancias de keyboard anteriormente seleccionado.

Test: samemaps

	Simple Logistic	1-NN	2-NN	3-NN	4-NN	NaiveBayes	SMO	J48	RandomForest	PART	REPTree
var1_nonorm_nobalanced_nostop	47.47	48.48	46.97	47.47	42.93	43.94	42.93	51.51	47.98	47.47	41.92

Test: othermaps

	Simple Logistic	**1-NN**	**2-NN**	**3-NN**	**4-NN**	**NaiveBayes**	**SMO**	**J48**	**RandomForest**	**PART**	**REPTree**
var1_nonorm_nobalanced_nostop	40.59	36.9	37.64	43.17	40.96	44.65	47.6	35.42	38.75	35.79	47.23

Resulta que la suposición que se tuvo en un inicio acerca de que los resultados de los modelos, para las instancias que fueron extraídas a partir de la IA programada en el tutorial 1, serían mejores, no se cumple. Esto puede deberse a muchas causas, desde datos con alto nivel de ruido, overfitting por falta de instancias de entrenamiento, underfitting, selección incorrecta de atributos, tratamiento incorrecto de atributos, etc. Aunque si se debe destacar la aparente paradoja en los resultados para los datos obtenidos por teclado frente a los recogidos por la IA, ya que a priori no parece que pudieran mostrar tal diferencia.

A falta de tiempo se ha decidido tomar como mejor modelo:

Algoritmo de ML	**Tipo de variación**	**Normalización**	**Balanceado**	**Limpieza**
RandomForest	6	No	No	Sí

Fase 3

El proceso de experimentación en este caso va a seguir un procedimiento similar al de la fase 2. Se va a seguir realizando en un inicio con los datos capturados por keyboard.

Primero se busca determinar qué atributos proporcionan suficiente información para el modelo de predicción. Para ello se tomarán todos los atributos y se hará una prueba para cada combinación entre normalización y limpieza de los atributos usando una regresión lineal.

- nonorm ≡ atributos no normalizados
- nostop ≡ data set limpio de instancias con valores Stop en los atributos action y pacmanDirec

LinearRegression	**test: samemaps**		**test: othermaps**	
	RMSE	RRSE	RMSE	RRSE
nonorm	58.74	34.3%	62.2	30.52%
norm	0.09	32.99%	0.08	30.15%
nonorm_nostop	64.15	32.31%	61.01	30.55%
norm_nostop	0.09	34.41%	0.08	32.76%

A partir de aquí, se escoge la mejor combinación de las probadas (resaltada en verde en la tabla superior) y se realizan 5 pruebas, cada una de estas corresponde a una variación de atributos como las de la fase 2. Las variaciones son las siguientes:

- **Variación 1 (todo):** pacmanDirec, pacmanXpos, pacmanYpos, LivingGhost1, LivingGhost2, LivingGhost3, LivingGhost4, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, ghost1Dist, ghost2Dist, ghost3Dist, ghost4Dist, currentScore, action, nextScore.

- **Variación 2 (sin livingGhost):** pacmanDirec, pacmanXpos, pacmanYpos, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, ghost1Dist, ghost2Dist, ghost3Dist, ghost4Dist, currentScore, action, nextScore.

- **Variación 3 (sin ghostDist):** pacmanDirec, pacmanXpos, pacmanYpos, LivingGhost1, LivingGhost2, LivingGhost3, LivingGhost4, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, currentScore, action, nextScore.

- **Variación 4** (**sin currentScore**)**:** pacmanDirec, pacmanXpos, pacmanYpos, LivingGhost1, LivingGhost2, LivingGhost3, LivingGhost4, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, ghost1Dist, ghost2Dist, ghost3Dist, ghost4Dist, action, nextScore.

- **Variación 5** (**sin livingGhost, ghostDist y currentScore**)**:** pacmanDirec, pacmanXpos, pacmanYpos, ghost1Xpos, ghost1Ypos, ghost2Xpos, ghost2Ypos, ghost3Xpos, ghost3Ypos, ghost4Xpos, ghost4Ypos, action, nextScore.

De tal manera, el conjunto de datos modificado que irá siendo probado dadas las variaciones mencionadas anteriormente y usando Linear Regression, será llamado como var<i>_norm, siendo <i> el número de la variación a experimentar.

LinearRegression	**test: samemaps**		**test: othermaps**	
	RMSE	RRSE	RMSE	RRSE
var1_norm	0.09	32.99%	0.08	30.15%
var2_norm	0.09	34.3%	0.07	28.24%
var3_norm	0.08	31.54%	0.07	27.67%
var4_norm	0.09	32.99%	0.08	30.15%
var5_norm	0.16	62.64%	0.15	58.49%

En este punto ya se tiene el conjunto de atributos que mejor resultados da para una regresión lineal (remarcado en verde en la tabla superior). A partir de aquí, se tomará la variación 3 y se evaluarán todos los algoritmos de predicción propuestos en función de los dos conjuntos de tipo test. Se harán evaluaciones para atributos normalizados y no normalizados.

Test: samemaps

	Bondad de ajuste	**Lin. Regression**	**MLP ∝=0.1 iter = 20**	**SMOReg**	**M5Rules**	**REPTree**	**RandomForest**
var3_nonorm	RMSE	59.17	82.11	60.47	60.17	63.30	69.95
	RRSE	**29.03%**	40.28%	**29.67%**	**29.53%**	31.06%	34.32%
var3_norm	RMSE	0.08	0.07	0.10	0.08	0.09	0.08
	RRSE	31.54%	**28.87%**	38.80%	31.53%	32.85%	30.22%

Test: othermaps

	Bondad de ajuste	Lin. Regression	MLP ∝=0.1 iter = 20	SMOReg	M5Rules	REPTree	RandomForest
var3_nonorm	RMSE	58.27	77.36	54.32	69.76	62.09	102.86
	RRSE	**28.76%**	38.18%	**26.81%**	34.43%	30.65%	50.77%
var3_norm	RMSE	0.07	0.08	0.07	0.08	0.07	0.1
	RRSE	**27.66%**	32.90%	30.39%	32.28%	**27.93%**	40.95%

Se han marcado en negrita los valores del Root Relative Squared Error < 30% (ya que no se ha conseguido bajar a menos del 20% después de todas las pruebas). Teniendo en cuenta dichos porcentajes, se realizarán las mismas pruebas sobre los datasets de instancias capturadas por la IA del tutorial 1 y se escogerá el modelo que proporcione mejores resultados para keyboard y tutorial 1.

Test: samemaps

	Bondad de ajuste	Lin. Regression	MLP ∝=0.1 iter = 20	SMOReg	M5Rules	REPTree	RandomForest
var3_nonorm_tutorial1	RMSE	61.73	80.04	67.05	77.93	87.35	62.89
	RRSE	**29.20%**	37.86%	31.72%	36.86%	41.32%	**29.74%**
var3_norm_tutorial1	RMSE	0.08	0.08	0.10	0.10	0.09	0.08
	RRSE	33.82%	33.84%	38.47%	41.58%	37.66%	32.24%

Test: othermaps

	Bondad de ajuste	Lin. Regression	MLP ∝=0.1 iter = 20	SMOReg	M5Rules	REPTree	RandomForest
var3_nonorm_tutorial1	RMSE	65.23	100.75	48.88	75.05	74.39	92.02
	RRSE	33.41%	51.62%	**25.04%**	38.45%	38.11%	47.14%
var3_norm_tutorial1	RMSE	0.09	0.14	0.06	0.08	0.09	0.12
	RRSE	39.93%	61.84%	**25.72%**	36.99%	42.83%	53.15%

Se observa que el algoritmo SMOReg entrenado con las instancias tomadas por la IA y evaluado con el conjunto de test 'othermaps' proporciona los valores más bajos de la raíz del error cuadrático relativo y de la raíz del error cuadrático medio. Este mismo algoritmo entrenado con instancias capturadas por teclado sin normalizar presenta un RMSE y un RRSE < 30%. Por lo tanto, el modelo seleccionado para predicción será el siguiente:

Algoritmo de ML	Tipo de variación	Normalización	Limpieza
SMOReg	3	No	No

Fase 4

La decisión de la elección del modelo de aprendizaje automático enfocado a clasificación ha consistido en un proceso tedioso de experimentación basada en preprocesado de datos y uso de diversos algoritmos posibles. Se han probado clasificadores aún no vistos profundamente en clase, como por ejemplo Random Forest o SMO.

En todo momento se ha buscado el algoritmo más capacitado para generalizar, siendo este entrenado con el dataset más adecuado posible. Para ello se han realizado distintas pruebas basadas en las variaciones comentadas en la fase 2. Se ha considerado que un modelo capaz de obtener porcentajes de clases correctamente clasificadas sobre ambos conjuntos test > 60% es suficientemente bueno para el total de instancias capturadas para el entrenamiento. Aún así, el modelo finalista seleccionado en la fase 2 no ha sido capaz de aportar la misma calidad de métricas usando los conjuntos de entrenamiento y test de las instancias de la IA del tutorial 1 en comparación con el mismo algoritmo entrenado con las instancias captadas por teclado.

Estas son las características del modelo de clasificación finalista:

Algoritmo de ML	**Tipo de variación**	**Normalización**	**Balanceado**	**Limpieza**
RandomForest	6	No	No	Sí

Esto quiere decir que el modelo se basa en un Random Forest entrenado con datos no normalizados, no balanceados y limpiados de instancias ruidosas de Stop para los atributos pacmanDirec y action.

Tras haber realizado una comparación entre el modelo entrenado con instancias por teclado y el de la IA del tutorial 1 se observa un mejor comportamiento en este primero. Esto era predecible desde el momento en el que se tenían presentes los valores de las métricas de evaluación de los modelos. Un ejemplo de esto se ve en el agente entrenado por teclado, el cuál es capaz de cumplir su objetivo prácticamente a la perfección siempre y cuando los fantasmas estén quietos. En el momento en que estos estén en modo aleatorio, el resultado empeora habiendo situaciones en las que el agente del pacman predice constantemente la misma dirección, “estancando” al comecocos en el mapa.

Al realizar estas mismas pruebas sobre los mapas con el agente entrenado por las instancias del tutorial 1, el mal comportamiento descrito anteriormente es más notorio. El pacman tiende a quedarse parado, ya que la clasificación da un valor de dirección a tomar que no se encuentra en las acciones legales del comecocos. Si se hubieran generado otros modelos basados en los demás algoritmos y datasets, el rendimiento habría sido peor dadas las métricas mostradas en las tablas de la fase 2.

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?**
   En mi caso particular hay diferencias notorias, ya sea por exceso de ruido en los datos, sobreaprendizaje por falta de una base de instancias contundente en cantidad ó una carencia de un preprocesado de datos más exhaustivo.

   Paradójicamente, el modelo que aprende de instancias provenientes del control por teclado ofrece una mejor calidad de clasificación de las acciones a tomar, en comparación con el modelo basado en instancias del agente automático.

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?**
   Siendo el objetivo tomar una decisión basada en el valor del Score en el siguiente turno, la clave sería tomar la dirección que mayor recompensa ofrezca. Esto podría derivar a trasladar el enfoque del problema hacia el aprendizaje por refuerzo, marcando una recompensa según las acciones posibles a tomar, pudiendo determinar si interesa tener en cuenta las recompensas “más cercanas en el tiempo” o “más lejanas”.

3. **¿Qué ventajas puede aportar predecir la puntuación respecto a la clasificación de la acción? Justifica tu respuesta.**
   Como se ha discutido en la pregunta anterior, esto podría resolver el problema del exceso de ruido en las instancias de aprendizaje, ya que el problema pasaría de ser aprendizaje supervisado a aprendizaje por refuerzo. El refuerzo que otorgan las recompensas basadas sería la puntuación predicha dada una dirección actual. Estas recompensas variarán según la dirección a tomar por el pacman, lo que equivaldría a la acción.

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 subido?**
   Puede llegar a resultar efectivo, ya que el modelo podría llegar a aprender de la relación entre descenso del Score y dirección previamente tomada. Sin embargo, para este problema en concreto, esto puede traer también la posibilidad de añadir ruido a las instancias. Se debe tener en cuenta que cada paso que da el pacman supone un descenso de -1 en el Score, sólo va a aumentar si se come un fantasma o un pacman.

   Para una solución más eficiente, se debería incorporar un sistema capaz de tener en cuenta también las probabilidades de reducción/aumento del Score más allá del tick siguiente, de tal

manera que la decisión esté optimizada y parezca tener un razonamiento más deliberativo y menos reactivo.

CONCLUSIONES

A pesar de la falta de instancias que proporcionan mayor capacidad de aprendizaje a los algoritmos utilizados durante la realización de esta práctica, considero que, dada la exhaustiva experimentación, los resultados obtenidos de las métricas de evaluación no son exageradamente malos, pero tampoco son excelentes.

Debo destacar mi descontento con los valores obtenidos con las instancias capturadas por la IA del tutorial 1. En un principio supuse que iba a ser información menos ruidosa y más precisa, sin embargo, por lo observado en la fase 2, esto no parece ser como se intuía a priori.

Con respecto a la fase 3, dedicada a la regresión y predicción de valores para el siguiente score, sucede lo mismo. El error relativo tiende a mantenerse en un rango de entre el 20% y el 30%. Habiendo trabajado ya con redes neuronales en situaciones diferentes y con datasets más completos que conseguían proporcionar valores de error inferiores al 2%, uno comprende que algo no está ayudando a que el modelo sea generalizable.

A pesar de haber planteado un procedimiento metódico para el preprocesado de datos, no era capaz de reducir el error por debajo del 20% en el caso de la fase de predicción. Para la situación que se proponía en la fase de clasificación, no llegó a incrementar el porcentaje de clases correctamente clasificadas por encima del 70%.

Con respecto a los problemas encontrados, estos los voy a dividir en cuatro puntos. El primero, corresponde a el exceso de ficheros que se generan para los conjuntos de entrenamiento y test de keyboard y tutorial 1 durante la experimentación.
El segundo deriva del primero, ya que no he encontrado manera de automatizar la experimentación en Weka, a pesar de que el Experimenter sea una herramienta muy cómoda, no me permitía introducir un conjunto externo de test para la evaluación. Era realmente tedioso y largo, probar algoritmo a algoritmo, para cada tipo de conjunto test y cada tipo de conjunto de entrenamiento, provocando en muchas ocasiones errores, forzando a repetir algunas de las pruebas.

El tercer punto corresponde a la problemática encontrada durante los intentos de instalación de javabridge en Windows aun aplicando lo establecido por la página oficial y foros. Finalmente opté por realizar un dual boot en el ordenador para establecer una partición basada en Debian. A partir de aquí, la instalación tuvo un ligero problema con respecto a una carpeta del open-jdk, pero finalmente se consiguió establecer la librería.

El cuarto y último punto ha sido uno de los más relevantes, y corresponde a la falta de un compañero con el que dividir tareas, contrastar ideas y paralelizar actividades de experimentación. Esto podría haber proporcionado más tiempo para una posible mejor elaboración de la práctica 1.