Universidad Carlos III de Madrid

Aprendizaje Automático

Grado en Ingeniería Informática

Grupo 83

Práctica 2: Aprendizaje basado en instancias

Autores:
Daniel Medina García
Alejandro Rodríguez Salamanca

5 de abril de 2016

Índice

1. Recogida de información **3**

2. Clustering **4**

3. Generación del agente automático **5**

4. Evaluación de los agentes **7**

5. Conclusiones **8**

Introducción

El presente documento contiene la memoria del trabajo realizado para esta segunda práctica de Aprendizaje Automático. En ella, el equipo ha utilizado el aprendizaje basado en instancias, haciendo uso de la técnica de *clustering* para poder implementar funciones de afinidad y agilizar así la clasificación. Continuamos así nuestra introducción al aprendizaje no supervisado, en oposición al aprendizaje supervisado visto en ejercicios anteriores.

1. Recogida de información

Tomamos como base la toma de datos de la anterior práctica, para construir sobre ella ligeras mejoras que en nuestra opinión optimizaban la información almacenada. Así, los siguientes datos fueron recopilados en tiempo de ejecución:

- `score` : atributo numérico que contiene la puntuación instantánea. Su rango es en principio ilimitado, si bien oscila en nuestros datos desde -30 hasta 600 jugando con el agente de teclado. Cabe esperar que los resultados obtenidos con el agente automático no sean tan buenos, por lo que consideramos razonable generalizar y considerar su rango de entre menos 1000 y 1000. Si bien esa cota superior es real (cada fantasma comido aporta 250 puntos y esta práctica usa tan solo 4 fantasmas), la inferior es un margen holgado en base a nuestros datos (para llegar a una puntuación tan baja el agente debería de no comer ningún fantasma y estar 1000 turnos sin comer ninguno, y esta situación tiene probabilidades despreciables de aparecer).

- `ghosts-living` : atributo numérico que lleva la cuenta de los fantasmas que quedan vivos en el momento actual. Su valor oscila entre 1 (sólo queda un fantasma por cazar) y 4 (situación inicial).

- `distance-ghost`$_i$: cuatro atributos numéricos que hacen de vector de distancias a los respectivos fantasmas en este turno.

- `prev-distance-ghost`$_i$: cuatro atributos numéricos que expanden la información desde el turno anterior para evaluar el acercamiento o distanciamiento a los fantasmas. Todas las distancias tienen como mínimo 1 y la diagonal del tablero como máximo.

- `pos<axis>` : coordenadas de PacMan, en forma de número entero. Sus valores se encuentran dentro del rango de los ejes X e Y del mapa de juego.

- `direction` : atributo enumerado que indica el anterior movimiento de PacMan, comprendido entre los cinco posibles valores *North, South, East, West* y *Stop*.

- `wall-<direction>` : 4 atributos enumerados en forma booleano que indican si hay o no un muro en las posiciones contiguas a PacMan.

- `move` : Acción que toma el agente en el turno actual, con posibles valores *North, South, East, West* y *Stop*, si bien esperamos que esta última no sea tomada nunca.

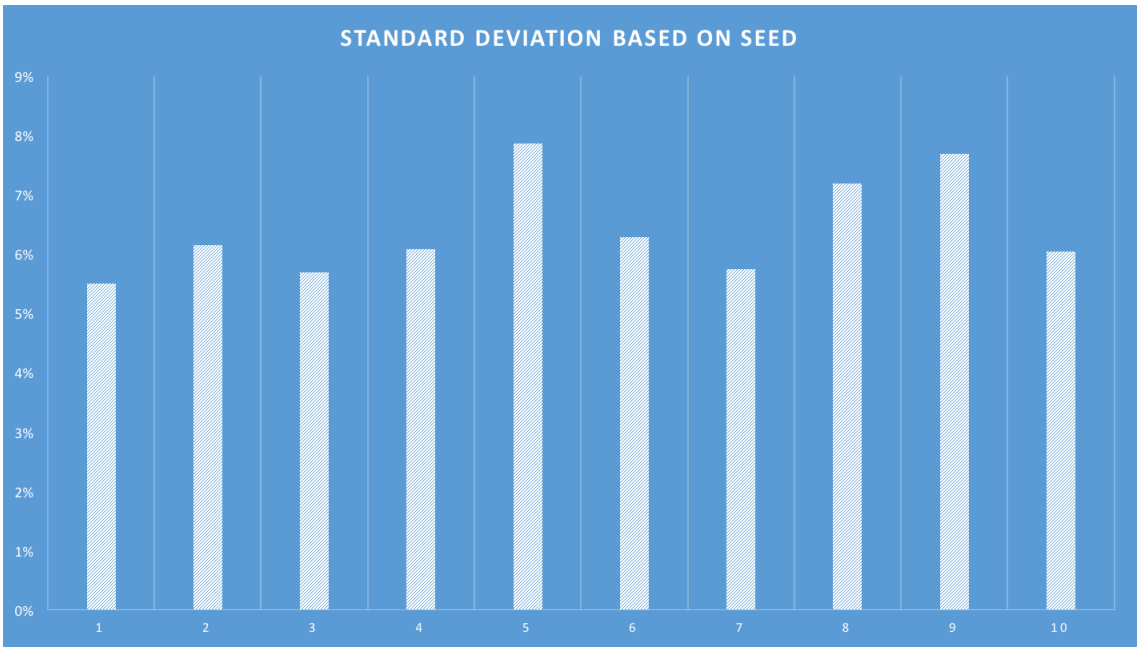
- En continuación de la parte de predicción de la *Práctica 1*, almacenamos seguidamente todos los atributos mencionados hasta ahora a cuatro turnos vista. Estos atributos pueden verse en el archivo como <atributo>N, indicando que es el atributo del turno futuro.

- `fx` : valor de la función de evaluación, que explicaremos más adelante.

2. Clustering

Tras probar todos los diferentes algoritmos de *clustering* ofrecidos por *Weka*, hicimos un primer filtro con aquellos que nos daban un número manejable de *clusters* (o se podía configurar dicho número) para evitar aquellos que generaban demasiados (menos de un 5 % de pertenencia) o insuficientes (menos de 4). Esta primera selección nos dejó con *Cobweb*, *EM*, *FarthestFirst* y *SimpleKMeans*. Comparando los algoritmos, buscamos dos propiedades: equilibrio entre los *clusters* y "estabilidad" entre ejecuciones al modificar los parámetros (i.e. semilla u otras constantes). Esta comparativa nos hizo decantarnos por *SimpleKMeans* y *EM*, pues los porcentajes de pertenencia a cada *cluster*eran más parecidos entre sí y distintas semillas resultaban en *clusters* de dimensiones similares.

Si bien los resultados eran parecidos entre estos dos algoritmos, el elevado coste en tiempo para elaborar el *clustering* con *EM* nos hizo decantarnos por *SimpleKMeans*. Mostramos a continuación la justificación de nuestra decisión, donde observamos el equilibrio conseguido con este algorimo de *clustering* y su estabilidad ante el cambio de la semilla. Cabe destacar que con los otros algoritmos encontramos variaciones muy superiores (e.g. entre 11 y 17 % con *FarthestFirst*, alcanzando el 15 % con *Cobweb*), alejadas de la media de 6 % obtenida con *SimpleKMeans*.



Para potenciar la eficacia de la clusterización, probamos a normalizar los datos. Sin embargo, los resultados obtenidos fueron los mismos. Como la normalización de los datos dificultaba la inclusión de nuevas instancias desde el wrapper de *Weka* para *Python* a un fichero ya normalizado, decidimos excluir este y otros filtros de preproceso de los datos. Para determinar si los movimientos tomados habían sido o no buenos, elaboramos una función cuyo incremento entre turnos supondría un rendimiento productivo. Esta función, que consta de tres sumandos con un coeficiente a modo de peso para evaluar distintos aspectos del agente, nos ayudará a descartar las instancias que no consideremos que ayuden para clusterizar correctamente:

$$f(x) = 0{,}5 * (\frac{distanceToClosest}{diagonal})^{-1} \ + \ 0{,}2 * (\frac{meanDistanceToGhosts}{diagonal})^{-1} \ + \ 0{,}3 * \frac{deadGhosts}{initialGhosts}$$

- En primer lugar, hacemos visible el acercamiento al fantasma más cercano normalizando la distancia a la que se encuentra PacMan de éste. Tomar la inversa de este dato nos asegura que un incremento en la función supone un movimiento acertado.
- Para evaluar el rendimiento a largo plazo, introducimos también la media de distancias normalizadas a los fantasmas. De nuevo, la inversa nos proporciona el dato que queremos incrementar.
- Por último, no podemos olvidarnos del objetivo principal del PacMan. Así, incluimos la cuenta de los fantasmas comidos en el último turno.

Al considerar de distinta imporancia los diferentes aspectos evaluados, incluimos los pesos de 0.5, 0.2 y 0.3 respectivamente a los sumandos previamente mencionados.

Las instancias pertenecientes a cada *cluster*se almacenaron en un array bidimensional: la primera dimensión indica el *cluster*y la segunda indica el índice de cada instancia dentro del cluster. A través del wrapper clusterizamos los datos de entrada al inicializar el agente automático, y según resulten en uno u otro *cluster*son añadidos a un u otro array.

En cuanto a la selección del *cluster*para cada instancia utilizamos, como anteriormente mencionamos, el algoritmo *SimpleKMeans* proporcionado por *Weka*, con 10 *clusters* y semilla igual a 4. El valor de la semilla es aleatorio y escogemos el cuatro como podíamos haber elegido otro, mientras que el número de *clusters* se ha escogido teniendo en cuenta que tenemos cuatro movimientos posibles, y varias situaciones posibles por las cuales tomar dichos movimientos, por lo que consideramos que 10 podía ser un número suficientemente significativo como para separar los datos.

3. Generación del agente automático

Una vez generados los *clusters* en los cuales separamos los datos entre los que clasificaremos las instancias nuevas, pasamos a lo que realmente integra la elección de la acción a tomar.

La función de similitud implementada asigna pesos a los diferentes atributos guardados de cada instancia para determinar un punto para cada instancia cuya distancia euclídea ponderada con otra instancia determinará cómo de afín le es. Repartimos los pesos según lo que consideramos "áreas de conocimiento", o grupos de atributos que contienen una misma información. De esta forma, agrupamos la puntuación, las distancias a los fantasmas, la posición a PacMan, la dirección y la existencia de muros alrededor, quedando de la siguiente forma:

$$similarityFunction(instance) = 0{,}3 * LivingGhosts \ + \ 0{,}1 * \sum_{i=0}^{livingGhosts} distanceToGhost_i \ +$$
$$0{,}2 * \sum_{i=0}^{gridDimension} coordinate_i \ + \ 0{,}1 * directionValue \ + \ 0{,}3 * \sum_{i=0}^{possibleWalls} thereIsWall_i$$

¿Por qué ha sido útil realizar *clustering* previa de las instancias?

El uso de *clusters* permite ahorrar bastante tiempo en comparaciones para la clasificación. Al tener *clusters* ya hechos, sólo se compara la instancia nueva con aquellas que pertenezcan al mismo *cluster*en lugar de con todo el set de entrenamiento, sabiendo que la instancia más cercana se hallará en dicho cluster.

¿Por qué es importante usar pocos atributos en técnicas de aprendizaje no supervisado?

El aprendizaje no supervisado busca redundancias entre las instancias que indiquen algún criterio con el que diferenciar grupos que permitan agrupar estos ejemplos en grupos. Encontrar redundancias es un proceso cuya dificultad crece de forma exponencial con respecto al número de atributos que se tienen en cuenta, por lo que minimizar el número de atributos con los que se hacen los *clusters* optimizará notablemente el proceso.

¿Qué ventaja tiene el uso del aprendizaje basado en instancias con respecto al visto en la práctica 1?

El aprendizaje no supervisado tiene como ventaja que no requiere datos previamente etiquetados sino que es el propio algoritmo el que busca el *grupo de datos* al que pertenece sin necesidad de saber los grupos que existen.

¿Consideras que el agente funcionaría mejor si se introdujesen más ejemplos? ¿Por qué?

El proceso de creación de *clusters* se basa en la búsqueda de redundancia entre instancias, por lo que un conjunto de entrenamiento más grande podría ayudar a formar *clusters* más informados. Cuantas más instancias tenga el proceso de clusterización más definidos estarán los grupos si los hay; si, por el contrario, los grupos no están definidos con un numero cuantioso de instacias, puede significar que los atributos recopilados no aporten la suficiente información como para separar las instancias como queremos y debamos buscar otro enfoque distinto para poder diferenciar instancias.

4. Evaluación de los agentes

5. Conclusiones

El modelo obtenido en esta práctica puede ser útil como parte del agente automático para el comecocos original, para la parte en la que PacMan se "comeüna pieza de comida y comienza a perseguir a los fantasmas, y podría modificarse para reutilizarlo invirtiendo las funciones de forma que "huyese"de ellos y buscase, en contraposición, las migas de pan que debe comerse. Al realizar la práctica hemos entendido campos donde podrían usarse técnicas de aprendizaje automático como el cálculo de rutas en tiempo real para un vehículo que se mueve por un entorno, desde un coche que se mueve por las calles de una ciudad hasta una aeronave que se mueve entre vientos cósmicos.

Problemas encontrados

como de costumbre en tareas de aprendizaje automático, los problemas surgidos durante el desarrollo de la *práctica 2* han solido estar relacionados con los "palos de ciego"necesarios para llegar a resultados concluyentes. bajo el lema de "prueba, error y aprendizaje", el equipo trabajó para afinar los resultados tanto como fuese razonable.

Comentarios personales

Nos ha parecido muy interesante cómo la combinación de diferentes técnicas da como resultado una búsqueda más eficiente de soluciones, como en esta práctica el procesamiento de asociación en *clusters* de los datos para reducir el dominio de búsqueda para la clasificación de una instancia. Este encuentro con el aprendizaje no supervisado nos ayudará a tener más herramientas a la hora de solucionar futuros problemas.