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

Introducción

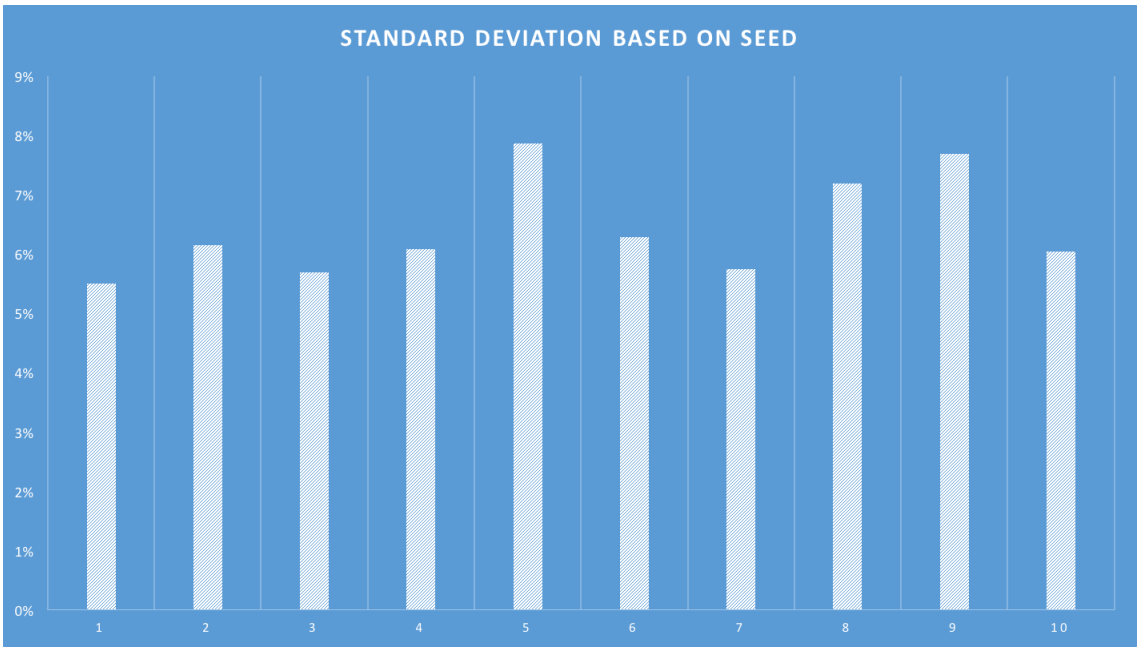
El presente documento contiene la memoria del trabajo realizado para esta segunda práctica de Aprendizaje Automático. En esta práctica el equipo ha utilizado el aprendizaje basado en instancias, haciendo uso de la técnica de *clustering* para poder implementar funciones de afinidad y agilizar así la clasificación.

1. Recogida de información

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 5). 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 con modificamiento en 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. 11 y 17 % con FarthestFirst, incluso 15 % con Coweb), 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.

- 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:

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

3.1. ¿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 compararemos 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.

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

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

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

Un conjunto de entrenamiento más grande podría ayudar a formar clusters más informados. Sin embargo, también haría más numerosas las instancias en cada cluster, ralentizando así el proceso de clasificación posterior. Este *drawback* podría contrarrestarse añadiendo un mayor número de clusters.

4. Evaluación de los agentes

5. Conclusiones

Problemas encontrados

Comentarios personales