uc3m | Universidad Carlos III de Madrid

PRÁCTICA 1

Aprendizaje Automático
30/03/2022

Marco Merola 100413665
Miguel Santana 100432505

ÍNDICE

INTRODUCCIÓN

¿Alguna vez se ha preguntado qué hace que un robot siga determinadas rutas para llegar de un lugar a otro? O quizás se haya preguntado ¿qué hace que un jugador del FIFA pueda seguir la pelota? o ¿Cómo puede este jugador describir movimientos en diagonal o línea recta según la posición de la pelota? Una parte de estas respuestas yace en los algoritmos de clasificación y aprendizaje automático. A través de la realización de esta práctica se pretende analizar e interpretar distintos algoritmos de clasificación y regresión en el juego de Pacman. Además se busca qué conjuntos de atributos y filtros pueden generar modelos que mejoren la actuación de Pacman en distintos mapas.

FASE 1: Extracción de información

Para la extracción de los datos, utilizamos un algoritmo que agrupa en instancias los distintos atributos que consideramos necesarios, como lo son la posición del pacman la de los fantasmas, el estado del fantasma(vivo o muerto), los movimientos legales del pacman, su distancia hacia cada fantasma, la puntuación (actual y del futuro) y la dirección que va a tomar. También se recolecta información de los muros en un área de 11x11, siendo el centro de esto el pacman. Para recolectar la puntuación del futuro, lo que hacemos es ir al siguiente turno y poner la puntuación de ese turno al punto anterior como puntuación futura. También nos damos cuenta que el último turno de una partida no tiene puntuación futura, por lo que hemos decidido eliminar esa instancia.

Para el training y los test con los mismos mapas usamos los mapas: training_1.lay, training_2.lay, training_3.lay, training_4.lay y training_5.lay.

Para los test en distintos mapas utilizamos los siguientes mapas: test_1.lay, test_2.lay, test_3.lay, test_4.lay y test_5.lay

FASE 2: Clasificación

Para la clasificación, en primer lugar se había seleccionado cierta cantidad de atributos, sin considerar información sobre el mapa. Inicialmente, almacenamos la posición del Pacman, las posibles acciones legales, el estado de cada fantasma, la posición de cada fantasma, la distancia hacia cada uno, la puntuación y la dirección. En un principio, utilizamos unas 500 instancias de entrenamiento y 200 para test. Analizando los resultados obtenidos, consideramos que una posible mejora era incluir más ejemplos de entrenamiento. Al realizar este cambio, se optó por incluir unas alrededor de 3000 instancias de entrenamiento y de test 500.

Al entrenar con estos datos, se encontró que los resultados sobre predicciones mejoraron ligeramente, en los propios mapas eran relativamente altos (alrededor de un 75% para varios algoritmos), sin embargo al entrenar con otros mapas se obtienen resultados más bajos. Esto se cree que se debe al sobreajuste a los mapas de entrenamiento y a que el modelo realmente estaba poco informado. También consideramos que esto se debía a que el score tenía rangos muy distintos y que de cierta forma afectaba a los resultados, se decidió normalizar este dato, sin embargo no se obtuvieron mejoras.

Para solventar los problemas anteriores, se ha considerado oportuno incluir información más precisa sobre cada mapa, con el fin de que los algoritmos de aprendizaje automático funcionen de mejor manera y pueda generalizar este tipo de información. En este caso, se ha decidido incluir una parte de área que recubre al Pacman, es decir indicar si las posiciones al norte, sur, este, oeste y diagonales para una distancia de cinco casillas tienen paredes o no. Existe la posibilidad de que estas posiciones no formen parte del área del mapa, esto se ha considerado equivalente a tener muros.

Algoritmos Seleccionados:

KNN: Se ha considerado interesante y oportuno probar un modelo con el algoritmo de clasificación KNN, puesto que es bastante simple, y además para este tipo de problemas, realmente se tienen pocos datos en comparación con un problema real de mayor escala donde el conjunto de entrenamiento puede ser de más de cien mil instancias. Este tipo de algoritmo consume mucha memoria, puesto que requiere el cálculo de la distancia euclídea pero para este dataset pequeño puede utilizarse sin problema. El algoritmo selecciona una cantidad de vecinos más cercanos a la instancia a clasificar, las cuales se encuentran en un mapa N dimensional y ya se sabe que pertenecen a una clase, por lo que según las característica de instancias similares a la del test, el algoritmo la clasificará

Random Forest: Dado que uno de los mayores problemas que se tienen en este tipo de problemas es que existen infinitos escenarios, puesto que los mapas pueden ser muy distintos entre sí, y además la posición de los fantasmas puede ser cambiante, por lo que el sobreajuste al conjunto de datos de entrenamiento es un problema bastante serio en estos casos. Precisamente, Random Forest es un algoritmo que genera distintos árboles de decisión que son independientes. Esta característica hace que el Random Forest sea adecuado para casos en los que pudiera existir sobreajuste.

Además, se probó con diversas semillas para el Random Forest, y variando este parámetro a través de la interfaz de Weka se logró mejorar el resultado obtenido.

Nota: Inicialmente, y antes de estudiar a mayor profundidad los algoritmos de clasificación, se consideró que el J48, como variante de ID3 podría ser una alternativa interesante ante este tipo de planteamientos, sin embargo al realizar las pruebas no fue así. De acuerdo con lo investigado, este algoritmo de clasificación suele presentar problemas de sobreajuste, por lo que al final se decidió no incluirlo a pesar de haber realizado pruebas con este.

Además, siguiendo las recomendaciones de la clase magistral se ha optado por probar diferentes conjuntos de atributos, siguiendo en parte la idea de la técnica Wrapper, sin embargo debido a la gran cantidad de atributos, realizar esta técnica completamente implica la existencia de muchas posibles combinaciones, por lo que se ha optado por realizar diversas pruebas con distintos conjuntos de atributos que se han considerado relevantes para el modelo.

Mediante la siguiente tabla se muestran los resultados de los modelos obtenidos según la interfaz de Weka:

Consideramos en primer lugar la clasificación sin incluir características sobre los muros dentro de los datos:

Automático

Samemaps:

Algoritmo	Porcentaje de instancias clasificadas correctamente
Ibk k=5	80.452%
Random Forest	86.4198%
Ibk k=3	79.8354%

Othermaps:

Algoritmo	Porcentaje de instancias clasificadas correctamente
Ibk k=7	56.682%
Random Forest	60.368%
Ibk k=5	53.9171%

Keyboard

Samemaps:

Algoritmo	Porcentaje de instancias clasificadas correctamente
Ibk k=1	69.0871%
Random Forest	76.971%
Ibk k=3	65.7676%

Othermaps:

Algoritmo	Porcentaje de instancias clasificadas correctamente
Ibk k=23	45.1537%
Random Forest	48.9362%
Ibk k=9	43.2624%

En segundo lugar pensamos en agregar los muros en un área de 11x11 donde pacman es el centro, y obtuvimos los siguientes resultados:

Automático

Othermaps:

Algoritmo	Porcentaje de instancias clasificadas correctamente
Ibk k=1	35.9447%
Random Forest	53.6866%
Ibk k=11	40.0922%

Samemaps:

Algoritmo	Porcentaje de instancias clasificadas correctamente
Ibk k=1	81.07%
Random Forest	86.214%
Ibk k=3	82.5103%

Keyboard

Othermaps:

Algoritmo	Porcentaje de instancias clasificadas correctamente
Ibk k=25	36.8794%
Random Forest	43.4988%
Ibk k=5	33.3333%

Samemaps:

Algoritmo	Porcentaje de instancias clasificadas correctamente
Ibk k=1	70.7469%
Random Forest	76.3485%
Ibk k=3	60.6929%

Gracias a estos resultados, consideramos que el mejor modelo es sin los atributos de los muros y con el agente automático.

FASE 3: Predicción

Random Forest: Se ha decidido considerar de nuevo el algoritmo Random Forest, puesto que este algoritmo permite la entrada de una gran cantidad de atributos, aspecto que nuestro conjunto de datos cumplía al almacenar información sobre el mapa. Además es considerado un buen modelo al emplear muchos árboles de regresión, lo que significa que suele tener alta precisión al predecir, a la vez que es un algoritmo que puede escalar bien, por lo que en caso de desear entrenar con un número mucho más alto de instancias o conjuntos más grandes de atributos esto no hubiese supuesto un problema en absoluto, y debido a la gran cantidad de cambios que se han realizado durante la práctica, consideramos útil emplear un algoritmo como el Random Forest para la regresión. Además, para el caso de la predicción de puntuación, creemos que este algoritmo puede trabajar de mejor manera ya que es tolerante frente a datos atípicos, por lo que si se tiene una instancia o un caso extraño este algoritmo seguirá actuando relativamente bien. Este último aspecto es muy positivo en esta tarea, pues el número de posibles escenarios es prácticamente infinito.

KNN: Una de las características más interesantes de este algoritmo y precisamente por la cual pensamos que incorporarlo en la regresión sería oportuno es que para “predecir” una salida el modelo utiliza la media sobre los valores locales cercanos indicados por el parámetro K, creemos que para este tipo de atributo la media puede ser una buena aproximación del valor de la puntuación.

A continuación se muestran los resultados obtenidos:

Considerando las instancias que no tienen información de las paredes:

Automático

Algoritmo	Error cuadrático medio
Ibk k=1	22.41
Random Forest	14.78
Ibk k=3	19.73

Keyboard

Algoritmo	Error cuadrático medio
Ibk k=1	23.66
Random Forest	15.67
Ibk k=3	20.47

FASE 4: Construcción de un agente automatico

Para esta tarea hemos seleccionado el modelo entrenado por el sin_muros/training_tutorial1_filter.arff con el algoritmo random forest ya que nos dan los mejores resultados con respecto a los demás algoritmos probados.

El agente creado por este modelo, no tiene un comportamiento muy bueno, ya que hay ocasiones donde entra en bucle, hasta que un fantasma se acerca lo suficiente para que este se mueva.

Comparando este agente con el que hemos realizado anteriormente da una clarísima conclusión, la cual es que este modelo no se les compara en lo más mínimo a los otros, ya que este no llega ni a terminar la partida, en cambio el agente automático del tutorial 1 es el más eficiente de todos porque consigue los caminos óptimos para comerse a los fantasmas, y por último el agente del teclado consigue comerse todos los fantasma pero la eficiencia depende de la persona que lo controla..

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?

Se cree que la diferencia principal se debe a que en el caso de un agente automático realmente eficaz y óptimo, siempre se encontrará la mejor solución, sin embargo algo que le ocurrió al equipo al controlar al agente es que se seguía a un fantasma que quizás estaba más lejos de otro, e incluso en muchas ocasiones hubo confusión con las teclas al realizar los movimientos del Pacman, y en algunos mapas más complejos errores humanos al seleccionar las rutas debido a la gran cantidad de paredes existentes, por lo que al almacenar datos en un fichero, se podrían generar instancias inapropiadas para realizar el entrenamiento.

Otra diferencia importante es que en caso de querer almacenar una gran cantidad de instancias, controlar un agente resulta incómodo y repetitivo para un humano. Por otra parte, un agente automático se puede ejecutar una gran cantidad de veces diseñando un script que ejecute comandos, sin necesidad de emplear humanos en esta fase, lo que puede reducir horas de trabajo y en proyectos más grandes esto se traduce en menos pérdida de dinero.

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?

Para transformar una tarea de regresión a clasificación, el valor que se intenta predecir (continuo por ser una regresión) debe ser transformado a discreto. Es decir, si por ejemplo se intenta predecir la calificación de un estudiante, en regresión lo común sería tener una salida con valores del 0 al 10. Sin embargo para transformar esto en una tarea de clasificación, esa salida debería transformarse en una con varios umbrales, como por ejemplo: excelente, bien, mal, entre otros. En el caso de la práctica, se deben asignar umbrales a las distintas puntuaciones y en función de eso pasaría de ser un valor numérico a tener cierta cantidad fija de posibles valores.

La aplicación práctica puede ser la de determinar si una partida es buena o no. También esto podría indicar si en una partida un jugador está “perdiendo” y en función de esa puntuación cambiar su modo de juego.

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

Para el caso de juegos como Pacman creemos que no da una ventaja significativa sobre la clasificación de una determinada acción en el juego, ya que este atributo no variará demasiado de iteración en iteración, es decir en mapas donde hay poca comida es un atributo que se mantiene casi constante y no es tan significativo como por ejemplo saber la posición actual de los fantasmas.

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?

Realmente creemos que incorporar este tipo de atributos no tiene mucho sentido si lo que se desea hacer es una mejora en el modelo de clasificación, puesto que realmente esto implicaría que la tarea que se está realizando no tiene mucho sentido, debido a que indicar si ha descendido en un determinado momento se acerca mucho a “guiar manualmente” las acciones del Pacman.

CONCLUSIONES

Esta práctica ha supuesto un reto interesante, pues ha permitido entender un poco mejor cómo es el trabajo que se debe realizar ante una tarea de aprendizaje automático. Asimismo, realizar esta práctica permite darse cuenta de la importancia de la extracción de datos en el aprendizaje automático, precisamente fue una de las fases que más tiempo y problemas implicó y que inicialmente se subestimó mucho y se creía que era la más simple de todas.

Nos hemos encontrado ante la complicación de recoger datos seleccionando un conjunto de atributos que inicialmente consideramos adecuado sin reflexionar mucho al respecto. Sin embargo, al terminar esta fase nos planteamos la posibilidad de que otros atributos como por ejemplo almacenar un parte del mapa podrían conducir a mejores resultados en los modelos, por lo que se tuvo que reiniciar toda la extracción de datos. Además, la otra dificultad que surgió fue la de que esos datos realmente fueran útiles, es decir que los mapas empleados para la fase de entrenamiento fueran representativos, diversos entre sí y que el Pacman se encontrase en distintas posiciones de partida. Tras considerar esto, se tuvo que repetir de nuevo la fase de extracción. Luego, surgió la duda de cuántas instancias deberíamos usar para entrenar. Al igual que muchas otras cosas, no hay una respuesta única para esto, por lo que probamos con distintas cantidades de instancias, hasta acertar en una que tenga en cierta medida equilibrados los resultados sobre SameMaps y OtherMaps, pues este tipo de casos se cree que tiende mucho a sufrir de overfitting. Esta situación nos ha permitido entender que una tarea de aprendizaje automático en la que se requieren millones de datos supone un esfuerzo muy grande y meticuloso en la fase de extracción de datos.

En cuanto a conclusiones técnicas sobre la tarea, encontramos que para este tipo de casos en los que hay tendencia hacia el sobreajuste, algoritmos como el Random Forest pueden maximizar las predicciones sobre el movimiento del Pacman. Por otra parte, se cree que realmente se necesita una mayor cantidad de mapas y muchas instancias de entrenamiento si se quiere dar con un mejor modelo que maximice la puntuación y haga predicciones más acertadas sobre los movimientos del Pacman. Además creemos que este tipo de tareas tiene aplicación en juegos importantes como el ajedrez, puesto que se podría determinar a través de un control sobre la puntuación (calculada mediante una función matemática) qué modo de juego emplear (ofensivo o defensivo) y de esta manera tomar decisiones al respecto.

Para finalizar, nos damos cuenta que los modelos de machine learning no son los mejores en cuanto a realizar una tarea como perseguir y capturar un objeto, ya que para que el modelo pueda aprender correctamente una tarea como esta se necesita un entrenamiento exhaustivo con muchos casos de prueba y una gran variedad de mapas.