UN ALGORITMO DE APRENDIZAJE AUTOMáTICO PARA EL JUEGO TIC-TAC-TOE ­

STATISTICAL TESTS TO COMPARE ALGORITHMS IN AUTOMATIC LEARNING STUDIES

Oraldo Jacinto Simón1,

Universidad Autónoma de Chihuahua (UACH), México, ory10887@gmail.com

Resumen: Tic-tac-toe (inglés americano), nudos y cruces (inglés británico), o Xs y Os es un juego de papel y lápiz para dos jugadores, X y O, que se turnan para marcar los espacios en una cuadrícula de 3 × 3. Cada una de las posiciones está representada con números del uno al nueve. Si los espacios son libres, puede seleccionar el número que corresponda en el rango de uno a nueve. Gana el primer jugador que lograr tres figuras consecutivas en línea recta. En el presente trabajo se presenta una variante de este juego donde la inteligencia de la máquina o computadora es lograda a través del aprendizaje automático usando como clasificador una red neuronal perceptrón multicapa. Para generar el conjunto de datos se desarrolla un algoritmo basado en reglas y se pone a jugar en contra de un jugador de manera aleatoria. Estos datos son procesados y en varias ejecuciones se construye único conjunto de datos donde se guardan todos los juegos con su mejor jugada. Se divide el conjunto de datos en dos subconjuntos, 25% de prueba, y 75% de entrenamiento. Se entrena el algoritmo y se serializa el clasificador para no tener que volver a entrenar a la hora de clasificar. Una vez entrenado el algoritmo se predice la mejor jugada hasta que uno de los jugadores pierda o gane. Para validar el algoritmo de aprendizaje automático se realizó una prueba de 10000, 25000, 50000 y 100000 partidas en el modo computadora(random) vs computadora (red neuronal perceptrón multicapa). Se presentan los resultados de dichas pruebas donde se puede ver con claridad que la computadora en su versión de aprendizaje automático obtiene mejores resultados.

Palabras Clave: tic-tac-toe, red neuronal multicapa perceptrón, aprendizaje automático

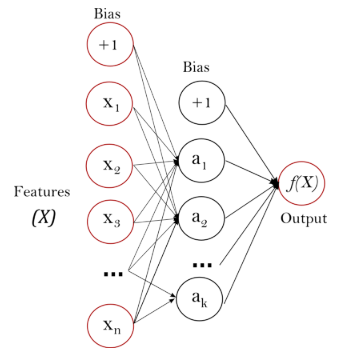
AbstracT: The Tic-tac-toe (American English), knots and crosses (British English), Xs and Os is a paper and pencil game for two players, X and O, who take turns marking the spaces in a 3 × 3 grid Each of the positions is represented by numbers from one to nine. If the spaces are free, you can select the corresponding number in the range of one to nine. Win the first player to achieve three consecutive figures online rec-ta. In the present work a variant of this game is presented where the intelligence of the machine or computation is achieved through automatic learning using as a classifier a multilayer perceptron neural network. To generate the data set, a rules-based algorithm is developed and played against a player randomly. These data are processed and in several executions are built in a single data set where all games are saved with their best play. The data set is divided into two subsets, 25% test, and 75% training. Find out the algorithm and classify the classifier so you do not have to re-train when classifying. Once the algorithm is trained, the best move is given that one of the players loses or wins. In order to validate the automatic learning algorithm, a test of 10000, 25000, 50000 and 100000 games are carried in the computer (random) vs computer mode (multilayer perceptron (MLP) is a deep, artificial neural network). The results of these tests are presented where it can be seen with clarity that the computer in its automatic learning version results better results.

**KeyWords**:tic-tac-toe, MLP, automatic learning

1. INTRODUCCIÓN

Tic-tac-toe (inglés americano), nudos y cruces (inglés británico), o Xs y Os es un juego de papel y lápiz para dos jugadores, X y O, que se turnan para marcar los espacios en una cuadrícula de 3 × 3. Cada una de las posiciones está representada con números del uno al nueve. Si los espacios son libres, puede seleccionar el número que corresponda en el rango de uno a nueve. Gana el primer jugador que lograr tres figuras consecutivas en línea recta.

En el presente trabajo se presenta una variante de este juego donde la inteligencia de la máquina o computadora es lograda a través del aprendizaje automático (AA) usando como clasificador una red neuronal perceptrón multicapa. El Perceptrón multicapa (MLP) es un algoritmo de aprendizaje supervisado que aprende una función al entrenar en un conjunto de datos, donde está el número de dimensiones para la entrada y el número de dimensiones para la salida. Dado un conjunto de características y un objetivo, puede aprender un aproximador de función no lineal para la clasificación o la regresión. Es diferente de la regresión logística, ya que, entre la entrada y la capa de salida, puede haber una o más capas no lineales, llamadas capas ocultas.[1]



Esta red tiene la capacidad para aprender modelos no lineales. Así como aprender modelos en tiempo real usando entrenamientos parciales.

Para generar el conjunto de datos se desarrolla un algoritmo basado en reglas y se pone a jugar en contra de un jugador random de manera aleatoria. A continuación, se explica como se realizó la generación de los datos.

Actividades:

Claramente explicado cómo generaron los datos.

Algoritmo y de entrenamiento y por qué usaron esos parámetros.

Resultados sobre el conjunto de datos de PRUEBA

Evidencia CLARA (matemáticamente visible) de que su algoritmo aprendió.

1. GeNERACIÓN de los datos

Después de que se ejecutan las corridas de las partidas de los jugadores *ComputadoraRandom* y *RulesAnalayzer* se guardan todas las movidas de cada uno de los juegos. La figura ‘X’ es representada con el número 1, la figura ‘0’ es representada con -1 y las casillas que no se llenan son representadas con cero. A continuación, se muestra en la variable \_copy que son representadas con los índices de cada una de las casillas del juego que son equivalentes a 9 posiciones. Y la última posición corresponde a la jugada idónea para ese estado del tablero.

**Ejemplo**:

**legenda**: (1=>X), (-1=>0) y (0=>empty)

\_**copy** = [1, 3, 5, 2, 9]

**output** = [-1, 1, 1, 0, -1, 0, 0, 0, 0, 8]

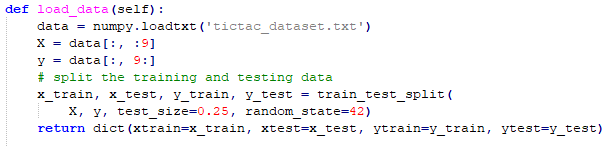
Para generar los datos se corrió el programa para un total de 10000 iteraciones en tres ocasiones en modo de entrenamiento. Los datos son salvados en los ficheros tictac\_dataset2.txt y tictac\_dataset3.txt que se encuentran en la raíz del proyecto.

Una vez generados estos datos se escalaron eliminando los duplicados y analizando los casos donde el jugador de la computadora random le ganaba. Al algoritmo de las reglas. Con todos estos datos se armó un único dataset de 6551 elementos. Este dataset contiene la inteligencia de todas las ejecuciones.

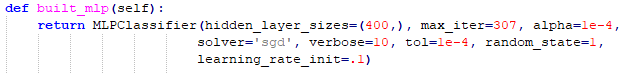
1. Algoritmo de entrenamiento

Se utilizó la clase MLPClassifier del paquete sklearn. Dicha clase implementa un algoritmo de perceptrón multicapa (MLP) que se entrena con Backpropagation [2] o propagando el error hacia atrás para ajustar los pesos de las neuronas.

*MLP entrena en dos matrices*: matriz X de tamaño (n\_samples, n\_features), que contiene las muestras de entrenamiento representadas como vectores de características de punto flotante; y una matriz y de tamaño (n\_samples,), que contiene los valores objetivo (etiquetas de clase) para las muestras de entrenamiento. Después de la adaptación (entrenamiento), el modelo puede predecir las etiquetas para nuevas muestras. En nuestro caso las etiquetas corresponden un AA supervisado donde el objetivo es predecir una jugada óptima en dependencia del estado del tablero. Primeramente, se dividen los datos en prueba y test.



Se realiza una búsqueda de la mejor configuración de parámetros para el clasificador seleccionado. El aumento de alpha puede arreglar una alta varianza (un signo de sobrealimentación) al fomentar pesos más pequeños, lo que da como resultado una gráfica de límite de decisión que aparece con menos curvaturas. La disminución del valor alfa puede corregir un alto sesgo (un signo de falta de adaptación) al fomentar un mayor peso.



1. Parámetros

**activation**: ´relu´, la función de unidad lineal rectificada, devuelve f (x) = max (0, x)

**alpha**: Alfa es un parámetro para el término de regularización, también conocido como término de penalización, que combate el sobreajuste ajustando el tamaño de las ponderaciones.

**batch\_size**: Tamaño de minibatches para optimizadores estocásticos. Cuando se establece en "auto", batch\_size = min (200, n\_samples).

**solver**: 'sgd’ gradiente estocástico descendente [3]

**verbose**: Imprimir mensajes de progreso a la salida estándar.

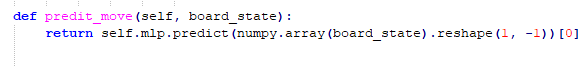
**tol**: Tolerancia para la optimización. Cuando la pérdida o el puntaje no mejoran al menos en tol para n iteraciones consecutivas, a menos que la tasa de aprendizaje se establezca en "adaptativa", se considera que se alcanza la convergencia y el entrenamiento se detiene.1

**random\_state:** Si int, random\_state es la semilla utilizada por el generador de números aleatorios

**learning\_rate\_init**: La tasa de aprendizaje inicial utilizada. Controla el tamaño de paso en la actualización de los pesos. Solo se utiliza cuando solver = 'sgd' o 'adam'. En nuestro caso usamos ´sgd´



Se entrena con un 75% y se obtiene la puntuación del clasificador con un 94% de score, para un 25 % del tamaño de los datos de prueba. Como podemos ver la puntuación del clasificador muestra una puntuación alta en cuanto a la clasificación de los datos de prueba.



La función predict\_move dado el estado del board predice una jugada óptima para la situación del tablero y devuelve un número del 1 al 9.

1. Pruebas

Una vez entrenado el algoritmo se predice la mejor jugada hasta que uno de los jugadores pierda o gane. Para validar el algoritmo de aprendizaje automático se realizó una prueba de 10000, 25000, 50000 y 100000 partidas en el modo computadora(random) vs Computadora (red neuronal perceptrón multicapa). Se presentan los resultados de dichas pruebas donde se puede ver con claridad que la computadora en su versión de aprendizaje automático obtiene mejores resultados.

Para ejecutar el juego ejecute el archivo Python game.py en python3. Sería pytho3 game.py.

A medida que se ejecuta el algoritmo se van generando los datasets para su posterior retroalimentación.

REFERENCIAS BIBLIOGRÁFICAS

[1] Scikit-learn.org, “Neural network models (supervised).” [Online]. Available: https://scikit-learn.org/dev/modules/neural\_networks\_supervised.html.

[2] N. Benvenuto and F. Piazza, “The backpropagation algorithm,” *IEEE Trans. Signal Process.*, vol. 40, no. 4, pp. 967–969, 1992.

[3] P. Tuttorial, “Descenso de Gradientes Estocástico ‘SGD,’” 2018. [Online]. Available: https://www.aprenderpython.net/descenso-gradientes-estocastico-sgd/.