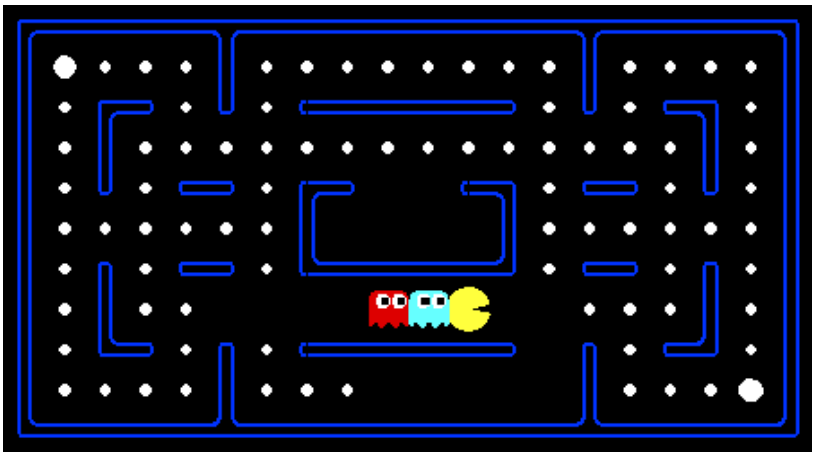
Proyecto 3: Clasificacion

Contenidos



¿Qué acción?

Introducción

En este proyecto, comenzarás por implementar las puertas lógocas AND, OR y XOR empleando perceptrones. Una vez que hayas entendido bien el uso del perceptrón sobre un conjunto de datos reducido, pasarás a diseñar dos clasificadores sobre un volumen mayor de datos: un clasificador perceptrón para reconocer dígitos y un clasificador perceptrón ligeramente modificado para la clonación conductual. Los datos se encuentran en el fichero datos.zip. Cuidado, depende de como lo descomprimas te generará una carpeta datos donde se encuentran las carpetas que necesitamos o directamente te generará las carpetas digitdata y pacmandata en el directorio actual. Una vez tengas ambas carpetas en el directorio actual, probarás el primer clasificador en un conjunto de imágenes de dígitos escritas a mano escaneadas y que se encuentran en la carpeta digitdata. El último en conjuntos de juegos pacman grabados de varios agentes. Incluso con funciones simples (foodCount), tus clasificadores podrán desempeñarse bastante bien en estas tareas cuando reciban suficientes datos de entrenamiento (estos datos se encontrarán en la carpeta pacmandata).

El reconocimiento óptico de caracteres (OCR) es la tarea de extraer texto de las fuentes de imagen. El conjunto de datos en el que ejecutará sus clasificadores es una colección de dígitos numéricos escritos a mano (0-9). Esta es una tecnología comercialmente útil, similar a la técnica utilizada por la oficina de correos de los Estados Unidos para enrutar el correo por códigos postales. Hay sistemas que pueden

funcionar con una precisión de clasificación superior al 99% (consulte LeNet-5 para ver un sistema de ejemplo en acción).

La clonación conductual es la tarea de aprender a copiar un comportamiento simplemente observando ejemplos de ese comportamiento. En este proyecto, utilizará esta idea para imitar a varios agentes pacman utilizando juegos grabados como ejemplos de entrenamiento. Tu agente ejecutará el clasificador en cada acción para intentar determinar qué acción tomaría el agente observado.

El código lo puedes encontrar en el fichero zip que se te proporcionan.

Ficheros con Datos	
`data.zip`	Contiene los datos sobre reconocimiento de dígitos
Ficheros que debeis editar	
`perceptron.py`	Contiene el programa en el que tendrás que añadir tu algoritmo perceptron
Ficheros que no deberías editar	
`classificationMethod.py`	Super clase Abstracta para los clasificadores que vas a emplear.
`samples.py`	I/O codigo para leer los datos.
`util.py`	Código con herramientas útiles.
`mostFrequent.py`	Un clasificador básico que clasifica todas las instancia con la clase más frecuente.

Evaluación: Las implementaciones de las puertas lógicas tendrán un valor de 8 puntos pero tu mismo podrás evaluar si has conseguido implementar y entrenar las puertas correctamente. Para evaluar el resto del proyecto (Q2, Q3 y Q4) se emplea el autograder. No cambies los nombres de ninguna clase ni método de los que se te proponen.

Pregunta 1 (8 puntos): Implementación de puertas lógicas empleando Perceptron

Se proveé de un Notebook en el que se te solicita que rellenes código donde encuentres el símbolo `pass # `. El nombre del Notebook es PerceptronLaborategiaIkasleak.ipynb y para ejecutarlo podrás hacerlo desde el Jupyter del anaconda-navigator u online empleando el Colab de Google. Para poder utilizar Google Colab, simplemente crea en tu drive una carpeta Colab Notebooks y descarga ahí el zip que está en eGela llamado PerceptronLaborategiaIkasleak.zip. Una vez descomprimido lo podréis ejecutar sobre Colab si no disponéis de anaconda-navigator el notebook

PerceptronLaborategiaIkasleak.ipynb.

Pregunta 2 (5 puntos): Perceptron

Se proveé de un esquema básico que encontraréis en el `perceptron.py`. Aquí deberíais de rellenar la función `train`.

Dada la lista de rasgos f, el perceptrón computa (predice) la clase y' en base a la multiplicación escalar $f\,w$. Formalmente, dado el vector de rasgos f el score obtenido para cada clase será (y'' representa una de las posibles clases o predicciones, mientras que y representa la verdadera clase y y' representa la que nuestro sistema calculará como su predicción favorita).

$$\text{score}(f,y'') = \sum_i f_i \cdot w_i^{y''}$$

Después se seleccionará la clase y'' que presente un mayor valor *score* (producto dot (escalar)).

Ajustando los pesos

En el perceptron multiclase básico, iremos recorriendo las instancias o ejemplos de entrenamiento, una instancia cada vez. Cuando estemos tratando la instancia (f,y), y como acabamos de explicar, seleccionaremos como nuestra predicción la clase (y'') con mayor *score*:

$$y' = \underset{y''}{argmax}\, score(f, y'')$$

Se compara y'con la verdadera clase y. Si $y'=y$, la instancia se ha clasificado correctamente, y por lo tanto no hay que llevar a cabo ninguna actualización. Por el contrario, si la predicción y'no se corresponde con y. Eso implica que wy'' donde y'' *es* y debería haber obtenido una puntuación f más alta, y por el contrario wy' debería haber obtenido una puntuación f más baja, y prevenir así que vuelva a ocurrir el mismo error en el futuro. Así que los pesos asociados a las dos clases implicadas se actualizan consecuentemente:

$wy = wy + f$

$wy' = wy' - f$

Nota: Podeís emplear la suma, subtraction, y multiplication ed la clase `Counter` en `util.py`, o implementarla vosotros.

Ejecuta tu código empleando:

```
python dataClassifier.py -c perceptron
```

Observaciones:

- El comando debería obtener una tasa de acierto entre un 40% y un 70%.
- Uno de los problemas del perceptrón es que es muy sensible a, por ejemplo, cuantas iteraciones se realizan sobre los ejemplos de entrenamiento, el orden en el que se presentan los ejemplos de entrenamiento (lo mejor es que sea aleatorio), la normalización de los rasgos..... El presente código está configurado para realizar 3 iteraciones. Podéis modificar el número de iteraciones a

través de la opción `-i iterations`. Emplea diferentes números de iteraciones y comprueba como varían los resultados. Si esto fuese un experimento real, deberíais de emplear la tasa de acierto sobre el conjunto de desarrollo para decidir cuando para de entrenar (cuantas iteraciones), pero para este ejercicio se ha simplificado la tarea.

Pregunta 3 (4 puntos): Clonación de comportamiento

Has construido dos tipos diferentes de clasificadores, un clasificador perceptrón y mira. Ahora emplearás una versión modificada de perceptron para aprender de los agentes pacman. En esta pregunta, completará los métodos de clasificación y capacitación en `perceptron_pacman.py`. Este código debe ser similar a los métodos que ha escrito en `perceptron.py`.

Para esta aplicación de clasificadores, los datos serán estados, y las etiquetas para un estado serán todas las acciones legales posibles desde ese estado. A diferencia del perceptrón para dígitos, todas las etiquetas comparten un solo vector de peso w, y las características extraídas son una función tanto del estado como de la posible etiqueta.

Para cada acción, calcule la puntuación de la siguiente manera:

$score(s,a)=w * f(s,a)$

Luego, el clasificador asigna cualquier etiqueta que reciba la puntuación más alta:

$$a' = \underset{a''}{argmax}\, score(f, a'')$$

Las actualizaciones de capacitación se producen de manera muy similar a la de los clasificadores estándar. En lugar de modificar dos vectores de peso separados en cada actualización, los pesos para las etiquetas reales y previstas, ambas actualizaciones se producen en los pesos compartidos de la siguiente manera:

$w=w+f(s,a)$

la acción correcta

$w=w-f(s,a')$

la acción predecida

Pregunta

Rellenar el método `train` en `perceptron_pacman.py`. Ejecutarlo llamando a:

```
python dataClassifier.py -c perceptron -d pacman
```

Este comando debería proporcionar validación y precisión de prueba 70%.

Pregunta 4 (4 puntos): Diseño de características de Pacman

En esta parte, definirás tus propios rasgos (features) para permitir que el agente clasificador clonado el comportamiento de los agentes observados. Hemos proporcionado varios agentes para que intentes copiar el comportamiento de:

StopAgent: un agente que solo se detiene

FoodAgent: Un agente que solo tiene como objetivo comer la comida, sin preocuparse por nada más en el medio ambiente.

SuicideAgent: Un agente que solo se mueve hacia el fantasma más cercano, sin importar si está asustado o no.

ContestAgent: un agente de personal de p2 que evita inteligentemente a los fantasmas, come cápsulas de energía y comida.

Hemos colocado archivos que contienen múltiples juegos grabados para cada agente en el directorio data / pacmandata. Cada agente tiene 15 juegos grabados y guardados para datos de entrenamiento y 10 juegos para validación y prueba.

Pregunta

Agrega nuevas funciones para la clonación por comportamiento en la función EnhancedPacmanFeatures en dataClassifier.py.

Al completar tus funciones, debes obtener al menos un 90% de precisión en ContestAgent y un 80% en cada uno de los otros 3 agentes proporcionados. Puedes probar esto directamente usando la opción --agentToClone <nombre del agente>, -g <nombre del agente> para dataClassifier.py:

python dataClassifier.py -c perceptron -d pacman -f -g ContestAgent -t 1000 -s 1000

Otras opciones útiles:

También te proporcionamos un nuevo ClassifierAgent, en pacmanAgents.py, que utiliza tu implementación de perceptron_pacman. Este agente toma datos de entrenamiento y, opcionalmente, validación, y realiza el paso de entrenamiento del clasificador tras la inicialización. Luego, cada vez que realiza una acción, ejecuta el clasificador entrenado en el estado y realiza la acción devuelta. Puedes ejecutar este agente con el siguiente comando:

python pacman.py -p ClassifierAgent --agentArgs trainingData = <ruta a los datos de entrenamiento>

También puedes usar la opción --agentToClone <Nombre del agente> para usar uno de los cuatro agentes especificados anteriormente para entrenar en:

python pacman.py -p ClassifierAgent --agentArgs agentToClone = <Nombre del agente>