Informe sobre la Ejecución del Algoritmo Genético en AWS

Ana Gutiérrez Mandingorra

1. Introducción

El presente documento detalla la configuración realizada en **Amazon SageMaker** para la correcta ejecución del algoritmo genético utilizando el archivo `launcher.py` de ejemplo. Se describen las modificaciones necesarias en el código fuente, los errores encontrados y las soluciones implementadas.

2. Configuración del Entorno en SageMaker

Para la ejecución del algoritmo en **Amazon SageMaker**, se utilizó un **estimador SKLearn** y se cambió la configuración y algunos parámetros del archivo `launcher.py` para adaptarlo a la ejecución del algoritmo genético propio:

- Se especificó el script de entrada como `genetic.py`.
- Se incluyó `requirements.txt` en la lista de dependencias, asegurando la instalación de las librerías adicionales necesarias para la ejecución del código.
- Se configuró `source_dir` para cargar todo el código directamente desde el directorio local, en lugar de utilizar un bucket de S3. Inicialmente, se intentó almacenar el código del algoritmo genético en un bucket de S3 y especificar su ruta en este archivo. No obstante, a pesar de que la ruta al bucket era correcta, se encontró un error aparentemente relacionado con la falta de permisos de ejecución sobre los archivos. Para evitar este problema, se optó por mantener el código en una carpeta local dentro de la sesión de JupyterLab.
- Se asignó la instancia `ml.t3.large`, dado que `ml.t3.medium` no proporcionaba los recursos adecuados para la ejecución eficiente del algoritmo.
- Se seleccionó la versión `0.23-1` de **SKLearn** para garantizar compatibilidad con el entorno.
- Se utilizó `py3` como versión de Python.

El código correspondiente a la configuración del estimador en `launcher.py` es el siguiente:

Código modificado en `launcher.py`

```
sklearn_estimator = SKLearn(
    entry_point="genetic.py",
    dependencies=["requirements.txt"],
    source_dir=".",
    role=role,
    instance_type="ml.t3.large",
    instance_count=1,
    framework_version="0.23-1",
    py_version="py3",
    sagemaker_session=sagemaker_session
)
```

3. Errores Encontrados y Soluciones Implementadas

Durante la ejecución inicial del script `launcher.py` en SageMaker, surgieron diversos errores relacionados con la importación de módulos, la sintaxis de anotaciones de tipo en Python y la compatibilidad con ciertas versiones de bibliotecas. A continuación, se detallan los principales errores encontrados y sus respectivas soluciones.

3.1. Error en la Importación de Módulos en `EdoAgent.py`

Al ejecutar `genetic.py`, se generaba el siguiente error:

```
ModuleNotFoundError: No module named 'helpers'
```

Este problema ocurría porque el archivo `EdoAgent.py` intentaba importar el módulo `helpers.py`, pero la ruta de búsqueda de Python no incluía el directorio de los agentes. Para solucionarlo, se añadieron las siguientes líneas al inicio del archivo:

Código añadido en `EdoAgent.py`

```
import sys
import os
sys.path.append(os.path.dirname(os.path.abspath(__file__)))
```

Esta modificación permite que Python reconozca la carpeta actual como parte del *path* de búsqueda de módulos.

3.2. Error en la Anotación de Tipos en `helpers.py`

En el archivo `helpers.py`, la función `get_development_card` contenía la siguiente anotación de tipo:

Código inicial en `helpers.py`

```
def get_development_card(hand, effect) -> int | None:
```

El uso de `int | None` para definir un tipo opcional de retorno solo está disponible a partir de **Python 3.10**. Dado que el entorno de SageMaker ejecutaba **Python 3.7**, esta sintaxis generaba un error de tipo:

```
TypeError: unsupported operand type(s) for |: 'type' and 'NoneType'
```

Para corregirlo, se reemplazó la anotación utilizando `Optional` del módulo `typing`, compatible con versiones anteriores de Python:

Código adaptado en `helpers.py`

```
from typing import Optional

def get_development_card(hand, effect) -> Optional[int]:
```

Esta solución mantiene la semántica original, asegurando compatibilidad con Python 3.7.

3.3. Error con `functools.cache` en `TristanAgent.py`

El archivo `TristanAgent.py` intentaba utilizar la función `cache` del módulo `functools`:

Código inicial en `TristanAgent.py`

```
from functools import cache
```

Sin embargo, `functools.cache` solo está disponible a partir de **Python 3.9**. Como el entorno de SageMaker ejecutaba **Python 3.7**, esto generaba el siguiente error:

```
ImportError: cannot import name 'cache' from 'functools'
```

Para solucionar este problema, se utilizó `functools.lru_cache`, que es compatible con versiones anteriores de Python:

Código adaptado en `TristanAgent.py`

```python
from functools import lru_cache

@lru_cache(maxsize=None)
```

Esta corrección permite utilizar la funcionalidad de `cache` sin requerir una versión más reciente de Python.

4. Conclusión

Tras la adaptación del código, se logró ejecutar exitosamente el algoritmo genético en **Amazon SageMaker** utilizando el script `launcher.py`. Gracias a estas modificaciones, el código pudo ejecutarse correctamente en el entorno de SageMaker, permitiendo la realización de distintos experimentos sobre el algoritmo genético de manera más eficiente.

Como parte de la entrega de la tarea *"Herramienta AWS Catan"*, se incluye este informe junto con el archivo `launcher.py` utilizado y la carpeta `Agents` adaptada con las modificaciones necesarias para su correcta ejecución. Por otro lado, los experimentos realizados y los resultados obtenidos en las ejecuciones se adjuntan en la tarea *"Algoritmo Genético para PyCatan en Cloud"*.