21 DE JUNIO DE 2024

PROYECTO FINAL

AZURE MACHINE LEARNING – NIVELES DE OBESIDAD

BRITANY TORRICO ROSAS
ESTUDIANTE DE CODIGO FACILITO
Bolivia

Contenido

Definición del Problema

Niveles de obesidad

Predicción de los niveles de obesidad en función de los hábitos alimentarios y la actividad física.

La obesidad, que provoca problemas físicos y mentales, es un problema de salud mundial con graves consecuencias. La prevalencia de la obesidad aumenta de forma constante, por lo que se necesitan nuevas investigaciones que examinen los factores que influyen en la obesidad y cómo predecir la aparición de la enfermedad en función de estos factores.

Acerca del conjunto de datos

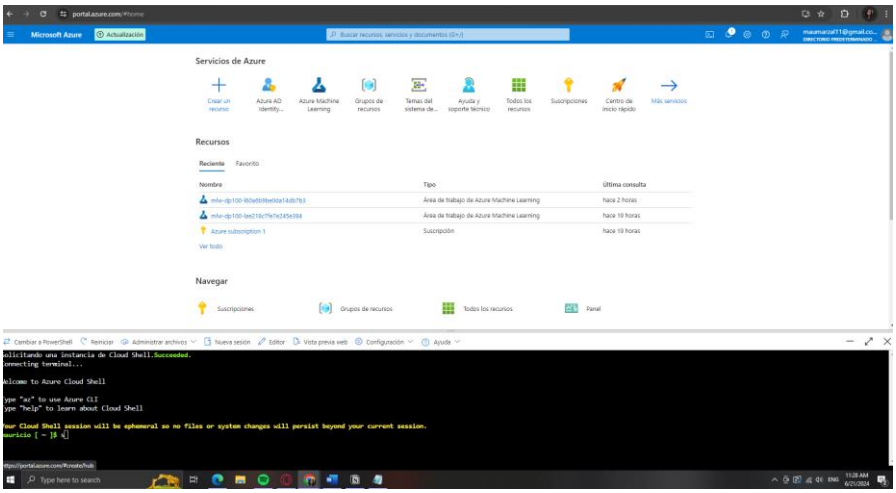
Este conjunto de datos incluye datos para la estimación de los niveles de obesidad en individuos de los países de México, Perú y Colombia, con base en sus hábitos alimentarios y condición física. Los datos contienen 17 atributos y 2111 registros, los registros están etiquetados con la variable de clase NObesity (Nivel de Obesidad), que permite la clasificación de los datos utilizando los valores de Peso Insuficiente, Peso Normal, Sobrepeso Nivel I, Sobrepeso Nivel II, Obesidad Tipo I, Obesidad Tipo II y Obesidad Tipo III. El 77% de los datos fueron generados de forma sintética utilizando la herramienta Weka y el filtro SMOTE, el 23% de los datos fueron recolectados directamente de los usuarios a través de una plataforma web.

- Género: Característica, Categórico, "Género"
- Edad: Característica, Continua, "Edad"
- Altura: Característica, Continua
- Peso: Característica Continua
- family_history_with_overweight: Característica, Binario, " ¿Algún miembro de la familia ha sufrido o sufre de sobrepeso? "
- FAVC : Característica, Binario, "¿Consumes alimentos altos en calorías con frecuencia?"
- FCVC : Característica, Entero, "¿Sueles comer verduras en tus comidas?"
- NCP : Característica, Continuo, "¿Cuántas comidas principales tienes al día?"
- CAEC : Característica, Categórico, "¿Consumes algún alimento entre comidas?"
- SMOKE : Característica, Binario, "¿Fumas?"
- CH2O: Característica, Continuo, "¿Cuánta agua bebes al día?"
- SCC: Característica, Binario, "¿Monitoreas las calorías que consumes diariamente?"
- FAF: Característica, Continuo, "¿Con qué frecuencia realizas actividad física?"
- TUE : Característica, Entero, "¿Cuánto tiempo utilizas dispositivos tecnológicos como celular, videojuegos, televisión, computadora y otros?"
- CALC: Característica, Categórico, "¿Con qué frecuencia bebe alcohol?"
- MTRANS: Característica, Categórico, "¿Qué transporte utiliza habitualmente?"
- NObeyesdad: Objetivo, Categórico, "Nivel de obesidad"

Los datos fueron utilizados de kaggle en el siguiente URL:
https://www.kaggle.com/datasets/fatemehmehrparvar/obesity-levels/data

Preparación del entorno:

1. Abrir el Cloud Shell de Azure



2. Ejecutar los siguientes comandos

   //eliminamos cualquier carpeta del proyecto por si ya fue creada anteriormente
   rm -r azure-ml-projects -f

   //clonamos el repositorio
   git clone https://github.com/BritanyTorrico/azure-ml-projects.git

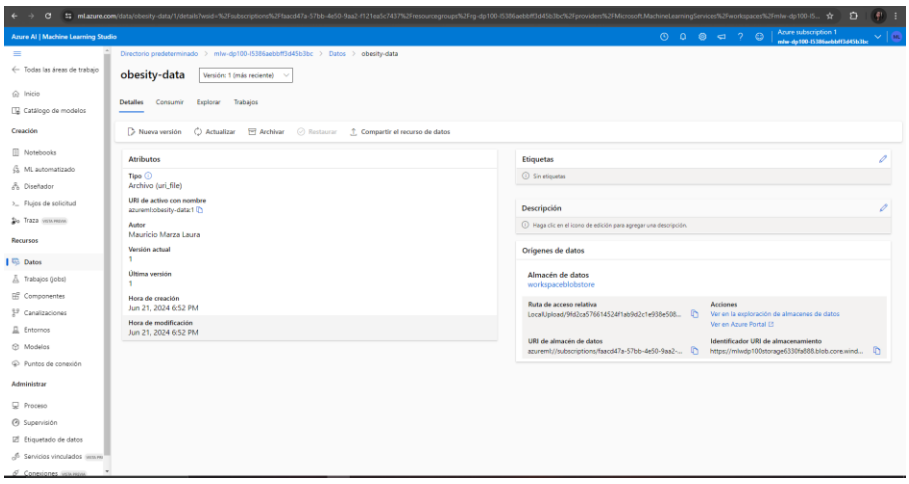
   // nos dirigimos a la carpeta del proyecto
   cd azure-ml-projects/projects/bri-proyect

   //asignamos permisos
   chmod +x setup.sh

   //ejecutamos el script que creara todos los recursos necesarios para azure machine
   ./setup.sh

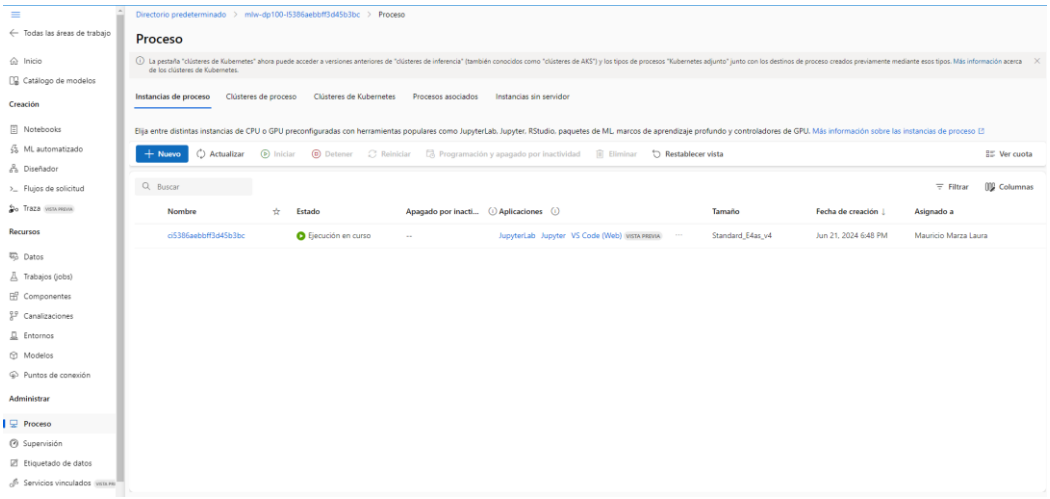
Ingresar al azure machine learning studio

1. Verificar que el los datos estén como un archivo URL

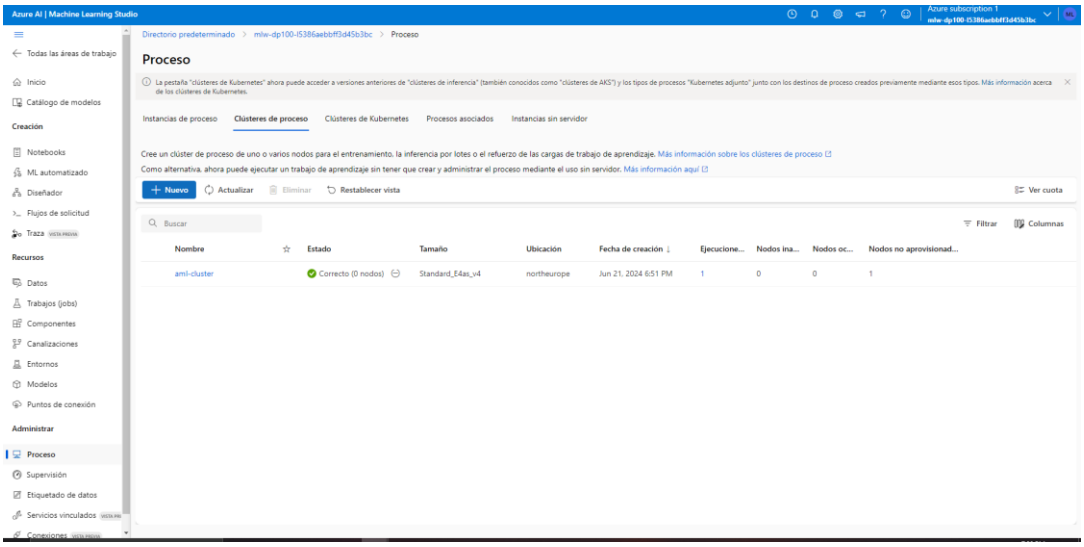


2. Verificar que se creó el proceso y clúster

-proceso

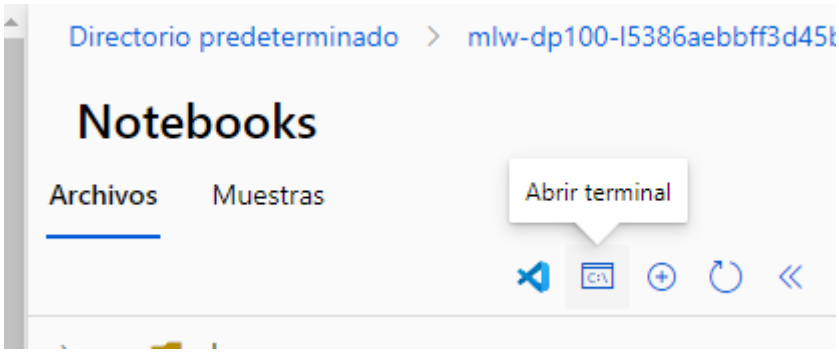


-Cluster



Azure Machine Learning Studio

En la terminal de nuestro Notebook



1. Abrir una terminal y ejecute los siguientes comandos

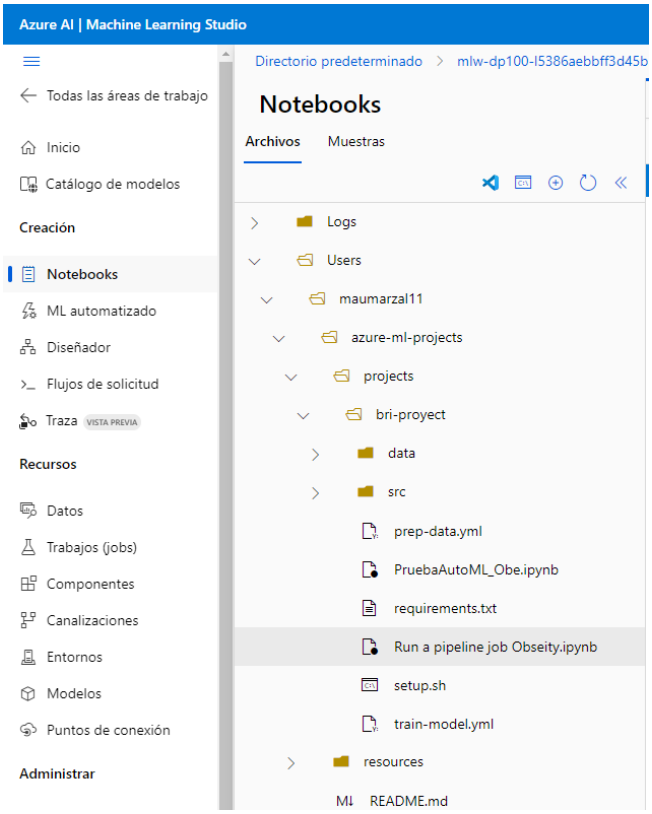
git clone https://github.com/BritanyTorrico/azure-ml-projects.git

cd azure-ml-projects/projects/bri-proyect

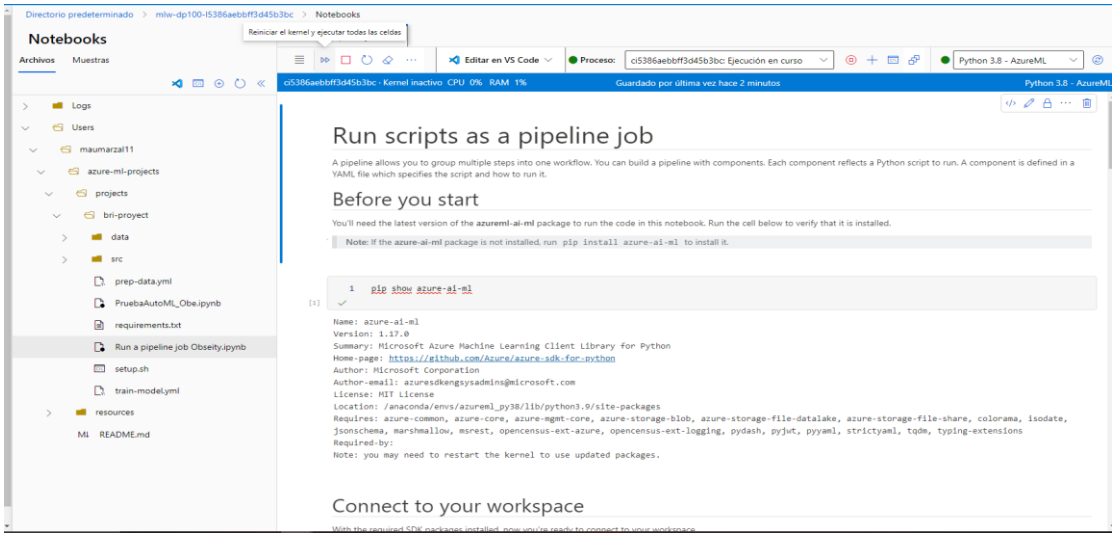
2. Instale todas las librerías necesarias con requirements.txt ejecutando el siguiente comando

pip install -r requirements.txt

3 . Entrar a Notebook y abrir la carpeta del proyecto



4. Ejecutar el archivo Run a pipeline job Obseity.ipynb



Explicación del codigo

Verificamos que `azure-ai-ml este instalado`

Run scripts as a pipeline job

A pipeline allows you to group multiple steps into one workflow. You can build a pipeline with components. Each component reflects a Python script to run. A component is defined in a YAML file which specifies the script and how to run it.

Before you start

You'll need the latest version of the **azureml-ai-ml** package to run the code in this notebook. Run the cell below to verify that it is installed.

> **Note:** If the **azure-ai-ml** package is not installed, run `pip install azure-ai-ml` to install it.

```
pip show azure-ai-ml
```

✓ 4 s

```
Name: azure-ai-ml
Version: 1.17.0
Summary: Microsoft Azure Machine Learning Client Library for Python
Home-page: https://github.com/Azure/azure-sdk-for-python
Author: Microsoft Corporation
Author-email: azuresdkengsysadmins@microsoft.com
License: MIT License
Location: /anaconda/envs/azureml_py38/lib/python3.9/site-packages
Requires: azure-common, azure-core, azure-mgmt-core, azure-storage-blob, azure-storage-file-datalake, azure-storage-file-share, colorama, isodate,
jsonschema, marshmallow, msrest, opencensus-ext-azure, opencensus-ext-logging, pydash, pyjwt, pyyaml, strictyaml, tqdm, typing-extensions
Required-by:
Note: you may need to restart the kernel to use updated packages.
```

Nos conectamos con nuestro workspace

Connect to your workspace

With the required SDK packages installed, now you're ready to connect to your workspace.

To connect to a workspace, we need identifier parameters - a subscription ID, resource group name, and workspace name. Since you're working with a compute instance, managed by Azure Machine Learning, you can use the default values to connect to the workspace.

```
from azure.identity import DefaultAzureCredential, InteractiveBrowserCredential
from azure.ai.ml import MLClient

try:
    credential = DefaultAzureCredential()
    # Check if given credential can get token successfully.
    credential.get_token("https://management.azure.com/.default")
except Exception as ex:
    # Fall back to InteractiveBrowserCredential in case DefaultAzureCredential not work
    credential = InteractiveBrowserCredential()
```

✓ 1 s

```
# Get a handle to workspace
ml_client = MLClient.from_config(credential=credential)
```

✓ <1 s

```
Found the config file in: /config.json
```

Aquí se crea una carpeta (src) para almacenar los scripts de Python (prep-data.py y train-model.py) que se utilizarán en los procesos de preparación de datos y entrenamiento del modelo, respectivamente. Se utiliza os.makedirs para asegurarse de que la carpeta exista.

Create the scripts

You'll build a pipeline with two steps:

1. **Prepare the data**: Fix missing data and normalize the data.
2. **Train the model**: Trains a decision tree classification model.

Run the following cells to create the **src** folder and the two scripts.

```
import os

# create a folder for the script files
script_folder = 'src'
os.makedirs(script_folder, exist_ok=True)
print(script_folder, 'folder created')
```

✓ <1 s

```
src folder created
```

- Preparación de los datos

El script prep-data.py se encarga de la preparación de los datos. Este script realiza varias tareas:

Lectura de datos: Lee datos desde un archivo CSV especificado por --input_data.

Limpieza de datos: Elimina las filas que contienen valores nulos.

Preprocesamiento de datos: Aplica transformaciones como imputación de valores faltantes y codificación de características numéricas y categóricas utilizando pipelines de sklearn. Luego, guarda el resultado en un archivo CSV (obesity.csv) en la ruta especificada por --output_data.

Preparar los datos

```
%%writefile $script_folder/prep-data.py
# importar librerías
import argparse
import pandas as pd
import numpy as np
from pathlib import Path
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer

def main(args):
    # leer datos
    df = get_data(args.input_data)

    cleaned_data = clean_data(df)

    prepared_data = preprocess_data(cleaned_data)

    output_df = prepared_data.to_csv((Path(args.output_data) / "obesity.csv"), index=False)

# función que lee los datos
def get_data(path):
    df = pd.read_csv(path)

    # Contar las filas e imprimir el resultado
    row_count = len(df)
    print('Preparando {} filas de datos'.format(row_count))

    return df

# función que elimina valores nulos
def clean_data(df):
    df = df.dropna()
```

Nota: Como estamos usando datos categóricos y numéricos es recomendable preprocesar la data

La función `preprocess_data` prepara un DataFrame `df` para el modelado, manejando características numéricas y categóricas. Utiliza pipelines para imputar valores faltantes y escalar características numéricas, y para codificar características categóricas. Luego combina estos procesamientos y devuelve un nuevo DataFrame con todas las características transformadas, listo para el análisis y entrenamiento de modelos

- Entrenando al modelo

una vez preparado los datos podemos entrenar al modelo utilizamos el modelo RandomForestClassifier por que es un modelo de clasificación que me va permitir tratar con mis diferentes tipos de datos que incluyen tanto caracteristicas numéricas como categóricas.

El script train-model.py se encarga del entrenamiento del modelo.

Realiza las siguientes operaciones:

Lectura de datos: Lee y concatena todos los archivos CSV de una carpeta especificada por --training_data.

División de datos: Divide los datos en conjuntos de entrenamiento y prueba.

Entrenamiento del modelo: Utiliza un clasificador RandomForestClassifier para entrenar un modelo de clasificación sobre los datos de entrenamiento.

Evaluación del modelo: Evalúa el modelo entrenado utilizando métricas como precisión, AUC-ROC, F1-score, precisión y recall.

En esta parte del entrenamiento registramos el modelo en mlflow con autologin e imprimimos algunas métricas en pantalla para la posterior revisión de los datos.

train model

```
%%writefile $script_folder/train-model.py
# importar librerías

import mlflow
import mlflow.sklearn
import argparse
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, precision_score, recall_score
import glob
import matplotlib.pyplot as plt

def main(args):
    # enable autologging
    mlflow.autolog()

    # leer datos
    df = get_data(args.training_data)

    # dividir datos
    X_train, X_test, y_train, y_test = split_data(df)
 #entrenar modelo
    model = train_model(X_train, y_train)

    # evaluar modelo
    eval_model(model, X_test, y_test)


# función que lee los datos
def get_data(data_path):
    all_files = glob.glob(data_path + "/*.csv")
    df = pd.concat((pd.read_csv(f) for f in all_files), sort=False)
```

- Definición de los componentes

En esta sección, se definen dos componentes de Azure ML utilizando archivos YAML (prep-data.yml y train-model.yml). Estos componentes encapsulan los scripts Python prep-data.py y train-model.py, respectivamente. Cada componente especifica los tipos de entradas y salidas que maneja, junto con el entorno de ejecución necesario para ejecutar los scripts.

prep-data.yml

```
%%writefile prep-data.yml
$schema: https://azuremlschemas.azureedge.net/latest/commandComponent.schema.json
name: prep_data
display_name: Prepare training data
version: 1
type: command
inputs:
  input_data:
    type: uri_file
outputs:
  output_data:
    type: uri_folder
code: ./src
environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
command: >-
  python prep-data.py
    --input_data ${{inputs.input_data}}
    --output_data ${{outputs.output_data}}
```

[7] ✓ <1 s

```
Overwriting prep-data.yml
```

train-model.yml

```
%%writefile train-model.yml
$schema: https://azuremlschemas.azureedge.net/latest/commandComponent.schema.json
name: train_model
display_name: Train a decision tree classifier model
version: 1
type: command
inputs:
  training_data:
    type: uri_folder
outputs:
  model_output:
    type: mlflow_model
code: ./src
environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
command: >-
  python train-model.py
    --training_data ${{inputs.training_data}}
    --model_output ${{outputs.model_output}}
```

[8] ✓ <1 s

```
Overwriting train-model.yml
```

- Cargar los componentes

Aquí se cargan los componentes definidos previamente (prep_data y train_model) utilizando load_component. Esto prepara los componentes para ser utilizados en la construcción de un pipeline de Azure ML.

Load the components

Now that you have defined each component, you can load the components by referring to the YAML files.

```
from azure.ai.ml import load_component
parent_dir = ""

prep_data = load_component(source=parent_dir + "./prep-data.yml")
train_decision_tree = load_component(source=parent_dir + "./train-model.yml")
```

[9] ✓ <1 s

- Creación del pipeline

Se define un pipeline de Azure ML llamado obesity_classification utilizando el decorador @pipeline. Este pipeline utiliza los componentes prep_data y train_model en secuencia, donde la salida del primero se convierte en la entrada del segundo. Esto permite automatizar y ejecutar de manera consistente el proceso completo de preparación de datos y entrenamiento de modelos.

Build the pipeline

After creating and loading the components, you can build the pipeline. You'll compose the two components into a pipeline. First, you'll want the `prep_data` component to run. The output of the first component should be the input of the second component `train_decision_tree`, which will train the model.

The `diabetes_classification` function represents the complete pipeline. The function expects one input variable: `pipeline_job_input`. A data asset was created during setup. You'll use the registered data asset as the pipeline input.

```
[10]
from azure.ai.ml import Input
from azure.ai.ml.constants import AssetTypes
from azure.ai.ml.dsl import pipeline

@pipeline()
def obesity_classification(pipeline_job_input):
    clean_data = prep_data(input_data=pipeline_job_input)
    train_model = train_decision_tree(training_data=clean_data.outputs.output_data)

    return {
        "pipeline_job_transformed_data": clean_data.outputs.output_data,
        "pipeline_job_trained_model": train_model.outputs.model_output,
    }

pipeline_job = obesity_classification(Input(type=AssetTypes.URI_FILE, path="azureml:obesity-data:1"))
```

✓ <1 s

Imprimir el pipeline_job

```
[11]
print(pipeline_job)
```

✓ <1 s

```
display_name: Train a decision tree classifier model
type: command
inputs:
  training_data:
    type: uri_folder
outputs:
  model_output:
    type: mlflow_model
command: 'python train-model.py  --training_data ${{inputs.training_data}}  --model_output
  ${{outputs.model_output}} '
environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
code:
/mnt/batch/tasks/shared/LS_root/mounts/clusters/ci5e062356045e41f3bf/code/Users/maumarzal11/azure-ml-
projects/projects/bri-proyect/src
is_deterministic: true
```

- Modificación de parámetros del pipeline

Se realizan modificaciones en los parámetros del pipeline antes de su ejecución. Esto incluye cambiar el modo de salida de algunos componentes a "upload" (indicando que los resultados deben ser cargados a un almacenamiento especificado), y establecer el entorno de cómputo (aml-cluster) y el datastore (workspaceblobstore) por defecto para el pipeline.

```
# change the output mode
pipeline_job.outputs.pipeline_job_transformed_data.mode = "upload"
pipeline_job.outputs.pipeline_job_trained_model.mode = "upload"
# set pipeline level compute
pipeline_job.settings.default_compute = "aml-cluster"
# set pipeline level datastore
pipeline_job.settings.default_datastore = "workspaceblobstore"

# print the pipeline job again to review the changes
print(pipeline_job)
```

[12] ✓ <1 s

```
type: command
inputs:
  training_data:
    type: uri_folder
outputs:
  model_output:
    type: mlflow_model
command: 'python train-model.py  --training_data ${{inputs.training_data}}  --model_output
  ${{outputs.model_output}} '
environment: azureml:AzureML-sklearn-0.24-ubuntu18.04-py37-cpu@latest
code:
/mnt/batch/tasks/shared/LS_root/mounts/clusters/ci5e062356045e41f3bf/code/Users/maumarzal11/azure-ml-
projects/projects/bri-proyect/src
is_deterministic: true
settings:
  default_datastore: azureml:workspaceblobstore
```

Guardamos el pipeline job con el metodo `create_or_update`

Submit the pipeline job

Finally, when you've built the pipeline and configured the pipeline job to run as required, you can submit the pipeline job:

```
# submit job to workspace
pipeline_job = ml_client.jobs.create_or_update(
    pipeline_job, experiment_name="pipeline_obesityBri"
)
pipeline_job
```

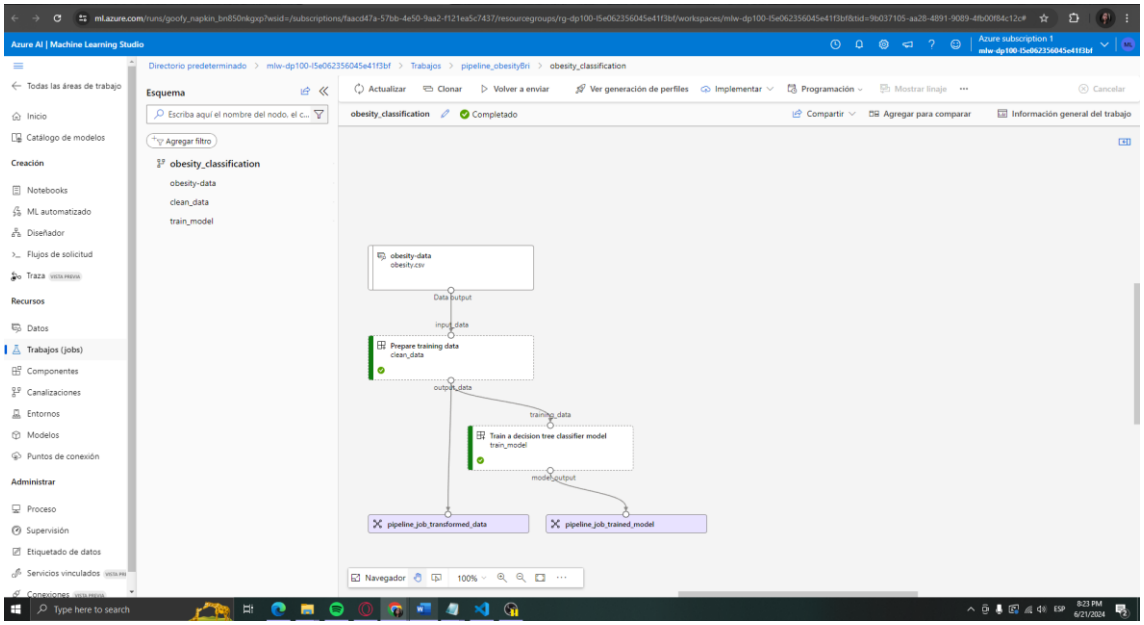
13] ✓ 10 s

```
Uploading src (0.01 MBs): 100%|██████████| 6596/6596 [00:00<00:00, 95611.37it/s]


pathOnCompute is not a known attribute of class <class
'azure.ai.ml._restclient.v2023_04_01_preview.models._models_py3.UriFolderJobOutput'> and will be ignored
pathOnCompute is not a known attribute of class <class
'azure.ai.ml._restclient.v2023_04_01_preview.models._models_py3.MLFlowModelJobOutput'> and will be ignored
```

Experiment	Name	Type	Status	Details Page
pipeline_obesityBri	goofy_napkin_bn850nkgxp	pipeline	NotStarted	Link to Azure Machine Learning studio

Nos dirigimos a la URL en pantalla



Informacion de entrenamiento de datos

Train a decision tree classifier model

Información general | Configuración | Resultados y registros | Métricas | Trabajos secundarios | Imágenes | Código | ...

Actualizar | Registrar modelo | Depurar y supervisar | Comparar (vista previa)

Propiedades

Estado
Completado

Fecha de creación
Jun 21, 2024 7:58 PM

Hora de inicio
Jun 21, 2024 7:58 PM

Duración
19.81 s

Duración de proceso
19.81 s

Nombre
899f0851-7ba8-4efe-8777-7352a7947c55

Comando
python train-model.py --training_data $AZUREML_DATAREFERENCE_training_data --model_output DatasetOutputConfig:model_output

Creado por
Mauricio Marza Laura

Tipo de trabajo
Paso de canalización

Entradas

Nombre de entrada: training_data
Recurso de datos: azureml_780f0625-941c-4f9c-89ee-02874979c0b7_output_data_output_data:1
URI del recurso: azureml:azureml_780f0625-941c-4f9c...

Salidas

Nombre de salida: mlflow_log_model_1879414278
Modelo: azureml_899f0851-7ba8-4efe-8777-7352a7947c55_output_mlflow_log_model_1879414278:1
URI del recurso: azureml:azureml_899f0851-7ba8-4efe...

Nombre de salida: model_output
Modelo: azureml_899f0851-7ba8-4efe-8777-7352a7947c55_output_model_output:1
URI del recurso: azureml:azureml_899f0851-7ba8-4efe...

Etiquetas

estimator_class : sklearn.ensemble._forest.RandomForestClassifier

estimator_name : RandomForestClassifier

Evaluación del Codigo

Si nos dirigimos al apartado de métricas podemos observar todas las métricas registradas

Métricas

1. Training Accuracy Score

Interpretación: La precisión (accuracy) es una métrica que indica la proporción de predicciones correctas (verdaderos positivos y verdaderos negativos) realizadas por el modelo sobre el conjunto de entrenamiento.

Valor: Un valor de 1 indica que todas las predicciones del modelo en el conjunto de entrenamiento fueron correctas, es decir, no hubo errores en las clasificaciones.

2. Training F1 Score

Interpretación: El F1-score es una medida que combina precisión y recall en una sola métrica. Es útil cuando las clases están desbalanceadas.

Valor: Un valor de 1 indica un F1-score perfecto, lo cual significa que el modelo tiene un buen equilibrio entre precisión y recall en el conjunto de entrenamiento.

3. Training Log Loss

Interpretación: Log Loss (o cross-entropy loss) es una medida de la precisión de un clasificador probabilístico. Cuanto más bajo sea el log loss, mejor será el modelo.

Valor: Un valor bajo (0.08620172 en tu caso) indica que el modelo tiene buenas predicciones de probabilidad sobre el conjunto de entrenamiento.

4. Training Precision Score

Interpretación: La precisión es una métrica que indica la proporción de predicciones positivas correctas sobre el total de predicciones positivas realizadas por el modelo.

Valor: Un valor de 1 indica que todas las predicciones positivas del modelo en el conjunto de entrenamiento fueron correctas.

5. Training Recall Score

Interpretación: El recall (o sensibilidad) es una métrica que indica la proporción de verdaderos positivos identificados correctamente por el modelo sobre el total de positivos reales en el conjunto de entrenamiento.

Valor: Un valor de 1 indica que el modelo identificó correctamente todos los casos positivos en el conjunto de entrenamiento.

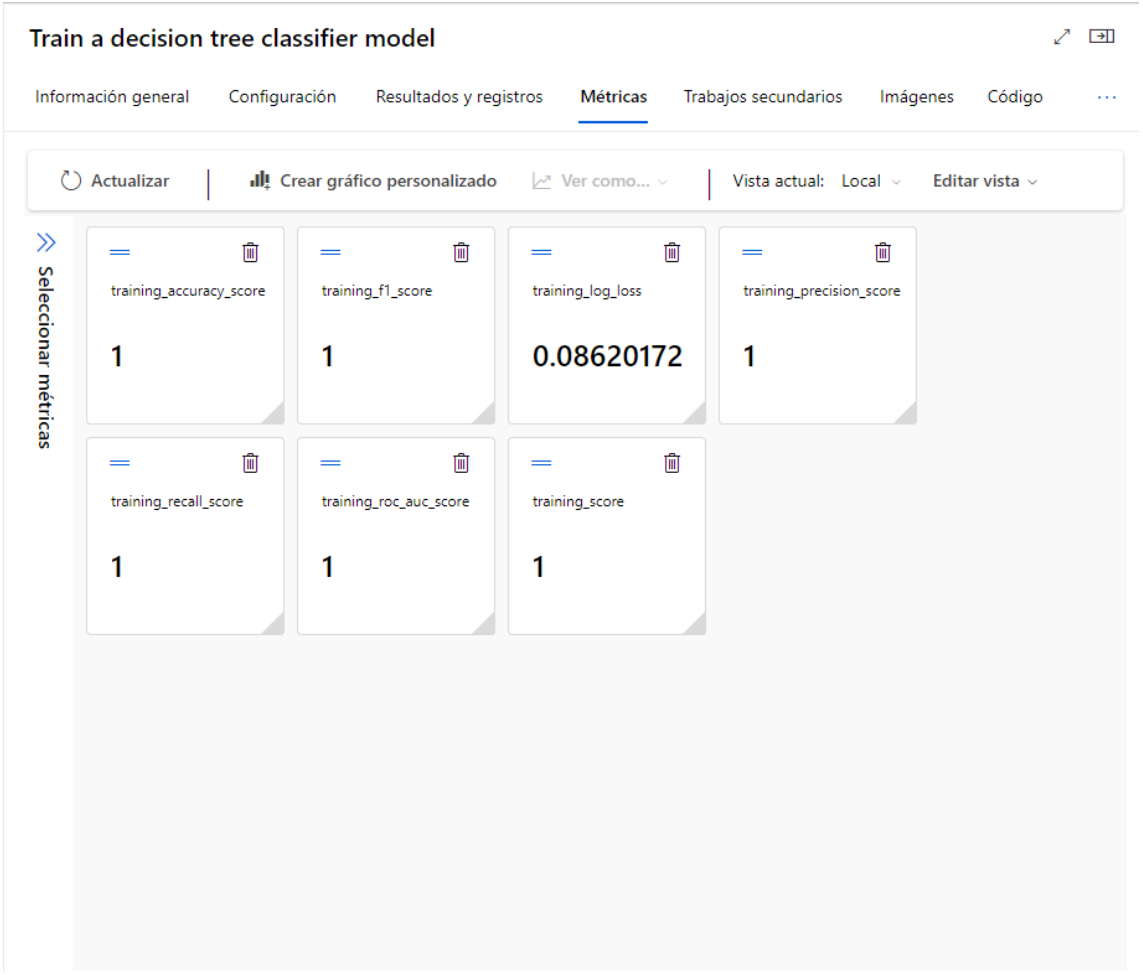
6. Training ROC AUC Score

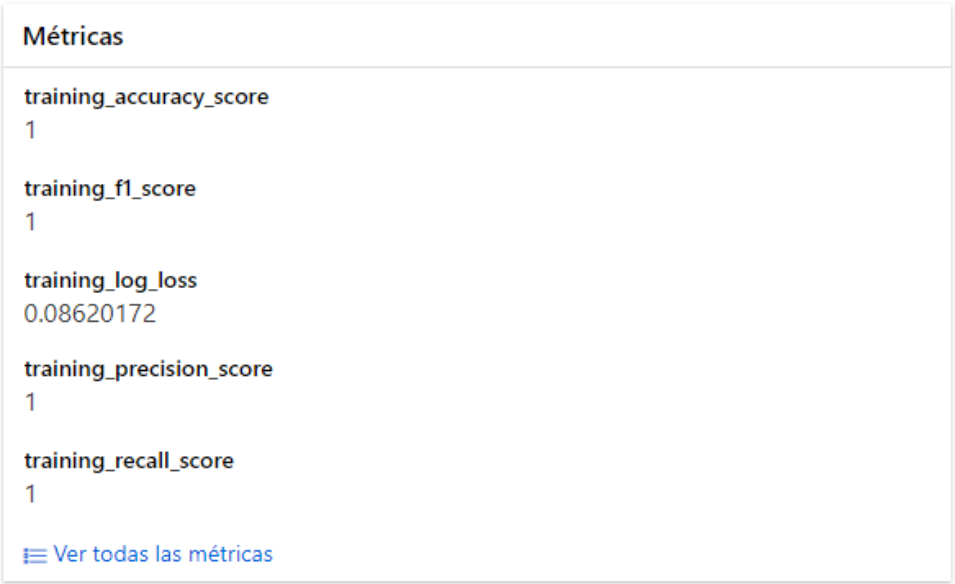
Interpretación: El área bajo la curva ROC (ROC AUC) es una métrica que evalúa la capacidad de discriminación del modelo. Representa la probabilidad de que el modelo clasifique correctamente un ejemplo positivo aleatorio más alto que un ejemplo negativo aleatorio.

Valor: Un valor de 1 indica un modelo perfecto que puede distinguir perfectamente entre las clases positivas y negativas en el conjunto de entrenamiento.

7. Training Score

Interpretación: Este puede variar según el contexto específico del modelo y la implementación. A menudo se refiere a una puntuación o medida compuesta del rendimiento del modelo en el conjunto de entrenamiento, que puede incluir diversas métricas combinadas o ponderadas.

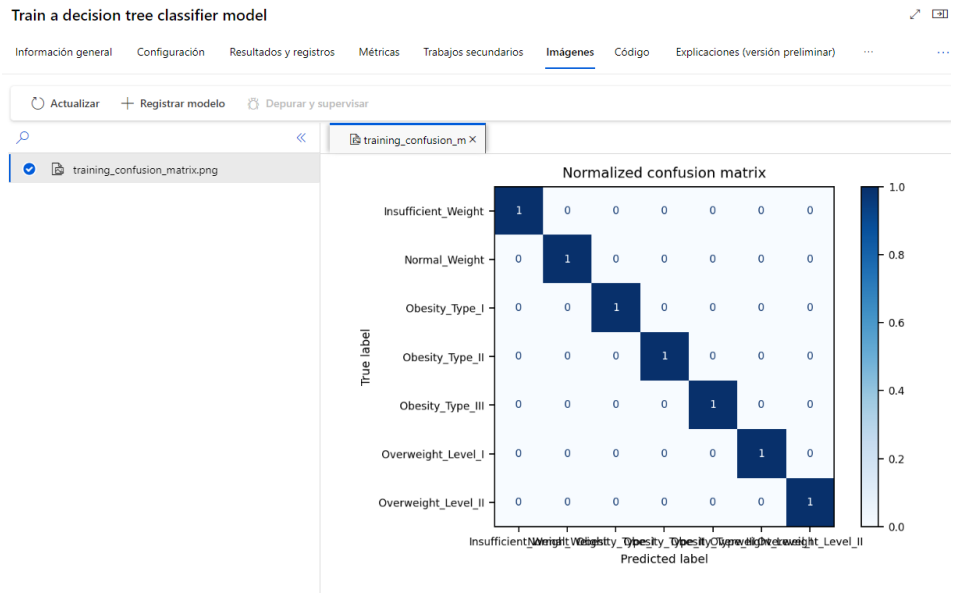




Matriz de confusión

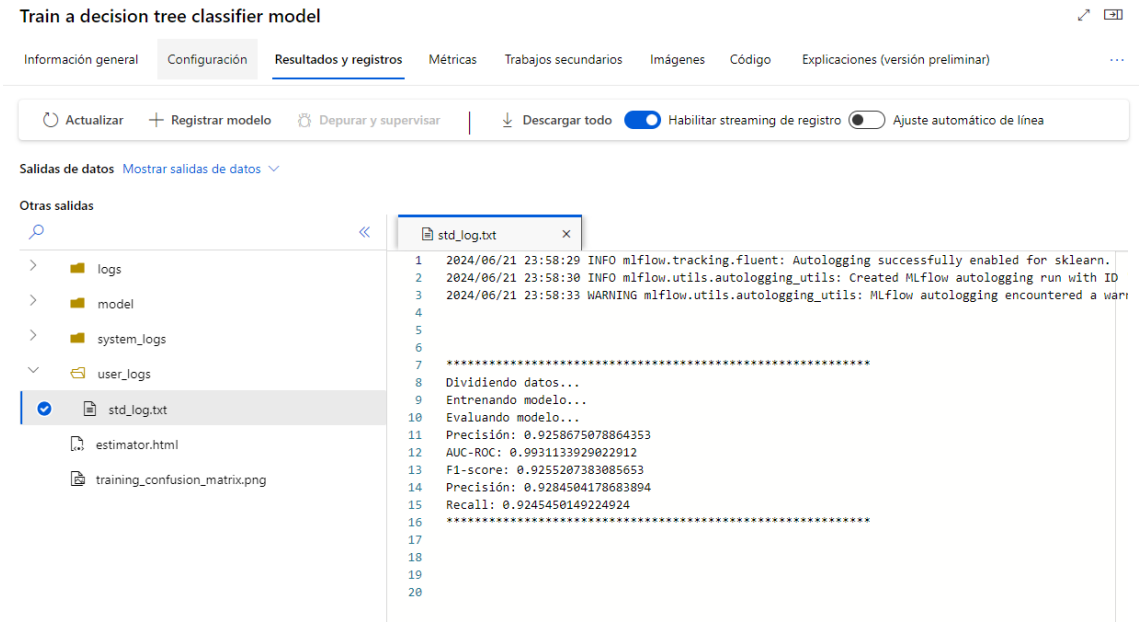
Las matrices de confusión proporcionan un aspecto visual de cómo un modelo de aprendizaje automático comete errores sistemáticos en sus predicciones para los modelos de clasificación.

La matriz de confusión de un modelo adecuado tendrá la mayoría de las muestras a lo largo de la diagonal



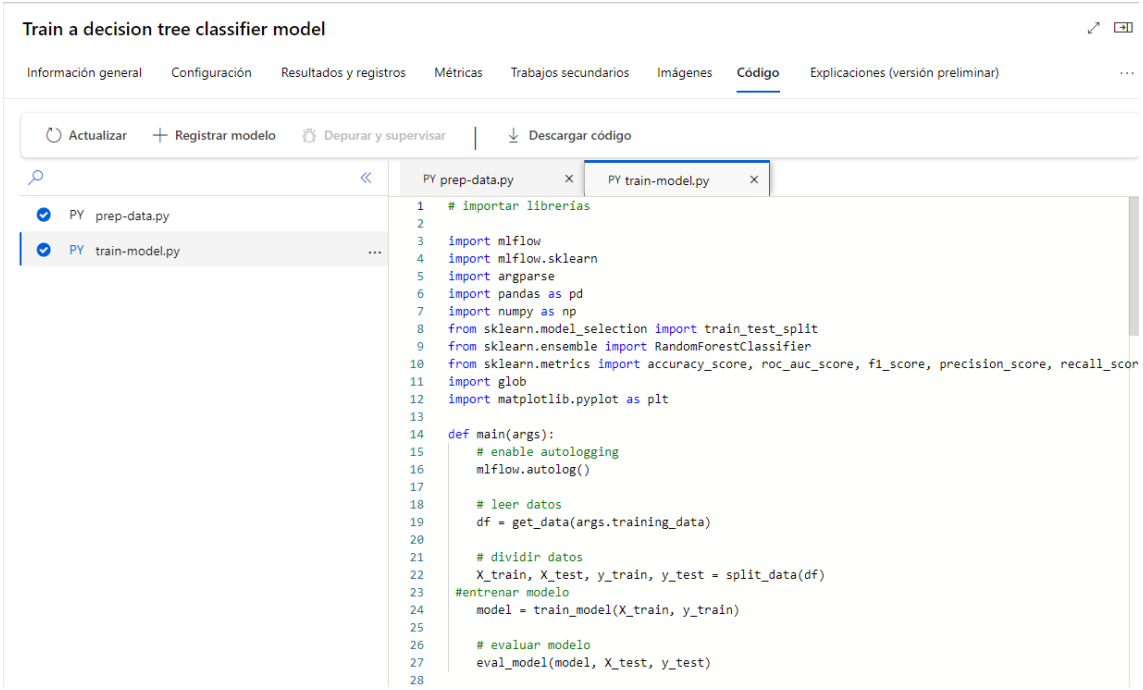
Resultados y registros

Si nos dirigimos a los resultados y registros podemos observar que se fueron imprimiendo nuestras métricas mientras se ejecuto el modelo



Codigo

En la seccion del codigo obetenemos los archivos de preparación y entrenamiento utilizados



MLFlow

Configuración del MLFlow en el modelo