Universidad Nacional de General Sarmiento

LABORATORIO DE CONSTRUCCIÓN DE SOFTWARE

Informe Progreso de Detección de Anomalías en Asistencia (Opción 4)

Alumnos:

- Tomás Altamirano
- Gonzalo Petosa

Elegimos trabajar con la opción número 4 debido a que nos pareció más adaptable a nuestra forma de trabajar con **Isolation Forest**.

Insolation Forest

El **Insolation Forest** es un modelo de machine learning diseñado específicamente para la detección de anomalías, es un modelo no supervisado que no requiere etiquetas en los datos para poder entrenarse. A diferencia de otros modelos que buscan clasificar o predecir una etiqueta, el objetivo de Isolation Forest es identificar puntos de datos que son anómalos o diferentes del resto

¿Cómo se utiliza este modelo?

- **Entrenamiento**: El modelo de **Isolation Forest** se entrena con un conjunto de datos que contiene tanto puntos de datos normales como posibles outliers (sin necesidad de etiquetas). El modelo construye un conjunto de árboles de decisiones binarias (de ahí el término "**Forest**") y utiliza particiones aleatorias de los datos para aislar las observaciones.

- **Predicción**: Después de entrenar el modelo, se puede utilizar para hacer predicciones sobre nuevos datos, y determinar si un punto de datos es "normal" o un "**outlier**". Los puntos que son más fáciles de aislar (requieren menos divisiones) se consideran **outliers**, mientras que los que son más

difíciles de aislar (requieren más divisiones) se consideran normales.

Características clave del modelo:

1. **Modelo no supervisado**: No necesita etiquetas para entrenar. Solo requiere los datos.

2. **Basado en árboles de decisión**: Utiliza múltiples árboles de decisión (un bosque) para aislar los puntos de datos.

3. **Eficiencia**: El modelo es eficiente y funciona bien en grandes volúmenes de datos.

4. **Robusto frente a grandes datasets**: Isolation Forest es especialmente útil cuando se trabaja con grandes conjuntos de datos con muchas características

LUEGO DE DESCARGAR EL CÓDIGO GIT

Antes de ejecutar el código, se debe de instalar mediante consola (CMD) las siguientes librerías de la siguiente manera:

```
pip install numpy pandas matplotlib scikit-learn openpyxl pip seaborn
```

Si está instalado, podemos comprobarlo al poner por cada librería:

```
pip show numpy
```

```
pip show pandas
```

```
pip show matplotlib
```

```
pip show scikit-learn
```

```
pip show openpyxl
```

```
pip show seaborn
```

De esta manera comprobamos si las librerías están instaladas en el equipo, mostrándonos información de la librería como su versión y otras características. En caso de que el programa en donde estamos escribiendo nuestro código estuviera abierto durante la instalación, hay que cerrarlo y volverlo a abrir para poder utilizar las librerías anteriormente nombradas.

Dataset Ficticio

Nuestro dataset consta de 30 empleados, para los cuales se crea una lista de 30 lugares a modo de representar la asistencia de los mismos durante 20 días, para esto se elige al azar un número que puede ser 0 o 1 siendo 0 si no asiste y 1 si asiste, nótese que hay un 10% de que sea 0 y un 90% de 1.

```
# Simulación de datos de 30 empleados y 20 días de registros
empleados = [f"Empleado_{i}" for i in range(1, 31)]  # Creación de los 30 empleados

dias = pd.date_range(start="2025-01-01", periods=20, freq='D')  # 20 días

asistencia_data = []  # Registro de asistencias
```

```
# Crear registros con '1' para asistencia y '0' para ausencia (con algunas ausencias repetidas para anomalías)
#Dataset
for empleado in empleados:
    asistencias = np.random.choice([0, 1], size=20, p= [0.2,0.8] ) # Genera un arreglo de tamaño 20 con 0 o 1 simulando la asistencia
    asistencia_data.append(asistencias)
```

Primero utilizamos una semilla de aleatoriedad para obtener un mismo resultado cada vez que testeábamos, luego comenzamos a simular los datos de los 30 empleados más los 20 días que estos deberían asistir. Lo siguiente que hicimos fué crear un **DataFrame** el cual es una tabla con los días con el valor de asistencia por empleado. Seguido de esto, usamos una expresión lambda para transformar los **0** en asistencias y los **1** en inasistencias ya que facilita el cálculo de la racha de ausencia.

```
# Convertir ausencias (0) a 1 y asistencias (1) a 0 para detectar anomalías
df_bin = df.applymap(lambda x: 1 if x == 0 else 0)
```

Entrenamiento y Predicción

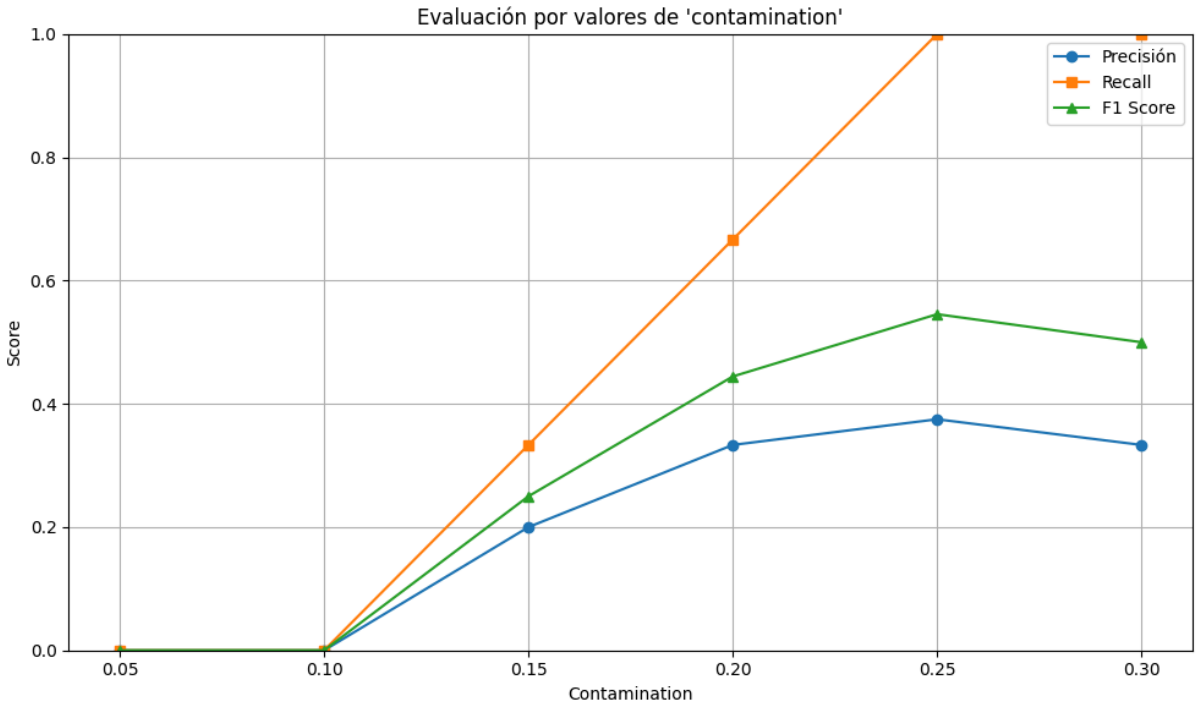
```
# Aplicar Isolation Forest para detectar anomalías
model = IsolationForest(contamination=0.2, random_state=42)  # El valor de contaminacion esta en 10% dado que esperamos ese porcentaje de ausentismo.
model.fit(df_bin)

# Predecir anomalías
anomalies = model.predict(df_bin)
```

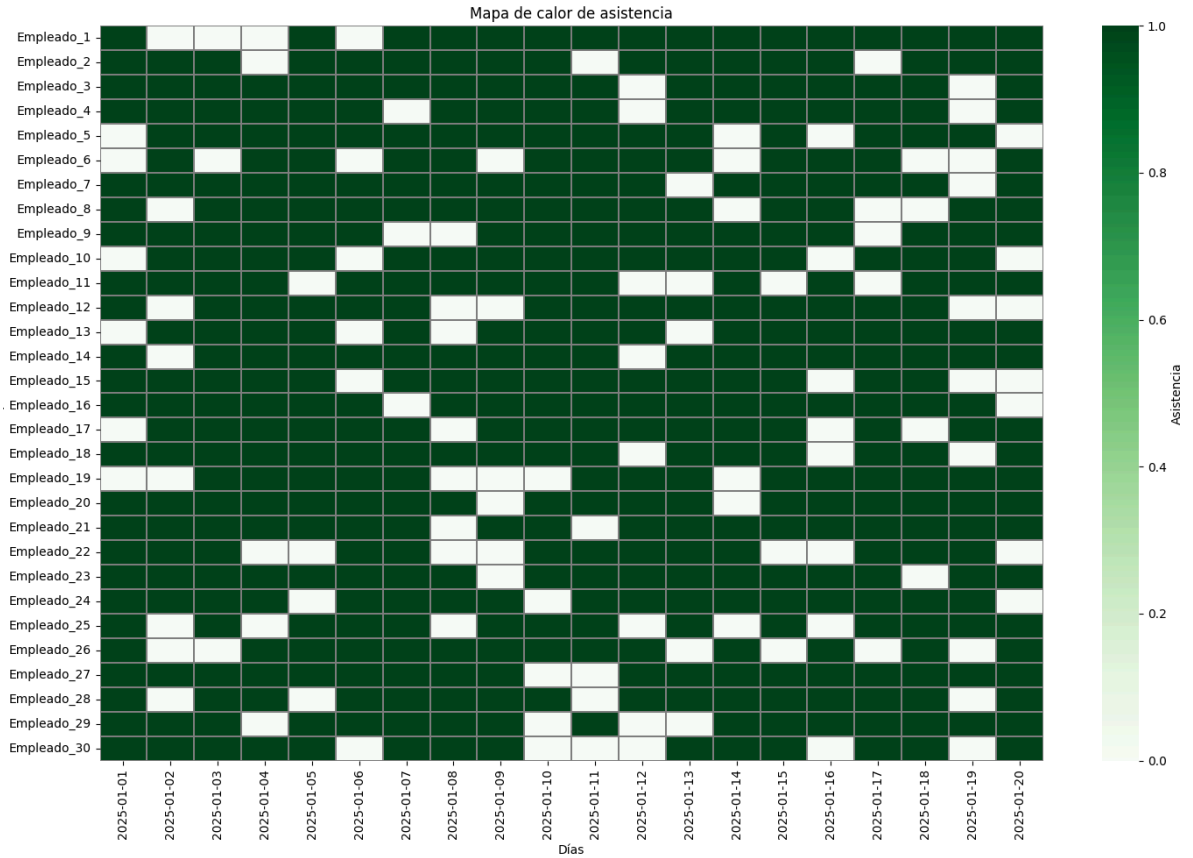
Entrenamiento (fit): El modelo aprende patrones normales de asistencia en tu dataset df_bin, que representa las ausencias (0 = asistió, 1 = se ausentó).
Predicción (predict): El modelo predice si cada fila del DataFrame representa un comportamiento anómalo (-1) o normal (1).

Luego aplicamos el **Insolation Forest** para comenzar a detectar las anomalías, dejando una contaminación en 0.2 ya que el porcentaje de ausentismo de un empleado ronda el 20%, y luego las predijimos en donde **1** es para normal y **-1** es para anomalía y por consiguiente creamos un DataFrame con estas predicciones, mostramos los resultados de estas para luego graficar sus resultados, y finalmente graficamos las asistencias de estos empleados, en dónde se nos muestra las anomalías que se registraron. Si bien el Insolation Forest es un modelo que **no puede ser entrenado**, este modelo al momento de ser puesto a prueba, es uno que puede detectar cuando hay anomalías de este tipo, aunque no con muy buena precisión.

PARTE GRÁFICA



En este primer gráfico adjunto se visualiza un gráfico con un eje X representando el nivel de contaminación, y un eje Y representando el score. También se muestran otras cosas como la **precisión** el cual representa que tan preciso fue el modelo al momento de seleccionar si un empleado estaba en estado normal o anómalo, en el ejemplo del gráfico, vemos que hubo una baja precisión por parte del modelo, ya que sólo el 33% de sus predicciones fueron acertadas. Luego tenemos el **recall** el cual representa la cantidad de anomalías verdaderas que el modelo encontró, es decir, cuando el recall esté en 1.0, quiere decir que encontró todas las anomalías verdaderas, más allá de haber seleccionado algunas de más. Y por último está el **F1 Score** que representa el puntaje de nuestro modelo en cuanto a la precisión y recall de este, es decir que nos dan un punto intermedio, por ejemplo en el caso del gráfico, tenemos una precisión del 33% y un recall del 67%, por lo tanto el F1 Score será de aproximadamente 0.42/1.



Este segundo gráfico nos muestra una tabla con los valores reales de inasistencias de los empleados, en donde tenemos a los empleados en el eje Y, y a los días en el eje X, cuando un recuadro está pintado de color verde, quiere decir que el empleado asistió, y cuando el recuadro está en blanco, quiere decir que el empleado faltó.

Nuestro código se encuentra en:

https://github.com/7oXaL/Plataforma-de-Gestion-de-Recursos-Humanos-con-ERP-y-CRM-con-Inteligencia-Artifical/blob/main/DataCet.py