Segunda entrega - Introducción a la Inteligencia Artificial: Predecir las emisiones de CO2 en Ruanda

Anderson Estiven Villa Sierra <estiven.villa@udea.edu.co>
Sebastián Ochoa González <sebastian.ochoa1@udea.edu.co>
Simón Alfredo Narváez Restrepo <simon.narvaez@udea.edu.co>

Inicialmente se procede a cargar el dataset sobre emisiones de CO2, tanto como los datos de prueba y entrenamiento mediante Kaggle CLI para acceder directamente a los datos de nuestra competencia desde nuestros notebooks, sólo cargando el json generado por cuenta de Kaggle, para poder acceder y ya tener total acceso sobre los datos sin tener que estar subiendo y bajando elementos a la máquina cada que se requiera probar.

Análisis para preparación y limpieza de datos

Antes de realizar un proceso de preparación y limpieza de datos, realizamos un análisis inicial para conocer el estado de los datos del dataset con el cual se trabajará. El primer proceso de análisis que se realiza es conocer la cantidad de valores faltantes en cada una de las columnas presentes, y se obtuvo lo siguiente.

```
SulphurDioxide_SO2_column_number_density          399
SulphurDioxide_SO2_column_number_density_amf      399
SulphurDioxide_SO2_slant_column_number_density    399
SulphurDioxide_cloud_fraction                     399
SulphurDioxide_sensor_azimuth_angle               399
                                                  ...
UvAerosolLayerHeight_solar_zenith_angle          3485
Cloud_sensor_zenith_angle                           1
Cloud_solar_azimuth_angle                           1
Cloud_solar_zenith_angle                            1
emission                                            1
Length: 63, dtype: int64
```

Lo cual nos indica que de 76 columnas que tenemos, hay 63 columnas que tienen por lo menos un dato faltante, como lo podemos ver en las columnas relacionadas con ‘SulphurDioxide’ las cuales tienen un total de 399 datos faltantes, y también notamos que la categoría “UvAerosolLayerHeight_solar_zenith_angle” tiene una cantidad muy considerable de datos faltantes, en este caso, 3485.

Por lo cual de este primer análisis de datos faltantes se decide extender un poco más a profundidad el análisis de datos faltantes en el dataset y se procede a extraer el porcentaje de datos faltantes por columnas para así tener más presente sobre qué columnas había una mayor ausencia de datos y se obtuvo lo siguiente.

```
[ ]   (d.isnull().sum() * 100/d.shape[0]).sort_values(ascending = False)

UvAerosolLayerHeight_aerosol_height          99.444466
UvAerosolLayerHeight_solar_zenith_angle      99.444466
UvAerosolLayerHeight_solar_azimuth_angle     99.444466
UvAerosolLayerHeight_sensor_azimuth_angle    99.444466
UvAerosolLayerHeight_aerosol_pressure        99.444466
                                               ...
latitude                                      0.000000
week_no                                       0.000000
year                                          0.000000
longitude                                     0.000000
emission                                      0.000000
Length: 76, dtype: float64
```

Donde se puede evidenciar que todas las características relacionadas al “UvAerosolLayerHeight” eran las que tenían un porcentaje mayor con respecto a las demás, en este caso tenían un porcentaje del 99.444466% lo que representa un gran volumen de datos faltantes.

Limpieza de datos

Con la premisa obtenida anteriormente, de que las características relacionadas con “UvAerosolLayerHeight” tienen un porcentaje tan alto de datos faltantes, se decide eliminar las columnas correspondientes a esas categorías, lo cual se realiza con los siguientes pasos expuestos:

- Listar las columnas del dataframe

```
d.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 79023 entries, 0 to 79022
Data columns (total 76 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   ID_LAT_LON_YEAR_WEEK                          79023 non-null  object
 1   latitude                                      79023 non-null  float64
 2   longitude                                     79023 non-null  float64
 3   year                                          79023 non-null  int64
 4   week_no                                       79023 non-null  int64
 5   SulphurDioxide_SO2_column_number_density      64414 non-null  float64
 6   SulphurDioxide_SO2_column_number_density_amf  64414 non-null  float64
 7   SulphurDioxide_SO2_slant_column_number_density 64414 non-null  float64
```

- Identificar las columnas correspondientes a eliminar, dónde se puede notar la ausencia de gran parte de los datos

```
56  Ozone_solar_zenith_angle                      78475 non-null  float64
57  UvAerosolLayerHeight_aerosol_height           439 non-null    float64
58  UvAerosolLayerHeight_aerosol_pressure         439 non-null    float64
59  UvAerosolLayerHeight_aerosol_optical_depth    439 non-null    float64
60  UvAerosolLayerHeight_sensor_zenith_angle      439 non-null    float64
61  UvAerosolLayerHeight_sensor_azimuth_angle     439 non-null    float64
62  UvAerosolLayerHeight_solar_azimuth_angle      439 non-null    float64
63  UvAerosolLayerHeight_solar_zenith_angle       439 non-null    float64
64  Cloud_cloud_fraction                          78530 non-null  float64
65  Cloud_cloud_top_pressure                      78539 non-null  float64
```

- Crear un nuevo dataframe, eliminando las columnas previamente identificadas (Basándose en los índices correspondientes)

```
[ ]    newData = d.drop(d.columns[57:64], axis=1)
```

Luego de eliminar las columnas con gran porcentaje de datos faltantes se toman dos decisiones para seguir con el proceso de limpieza de datos, en este caso con respecto a los datos faltantes en las otras columnas que no representan un porcentaje tan alto. Frente a este escenario se tomaron dos decisiones.

- Llenar los datos faltante con la media
- Eliminar las filas con datos faltantes

Para ambos casos se probaron dos modelos de predicción distinto con el fin de analizar cómo responden los modelos con estas dos decisiones tomadas con respecto a los datos faltantes.

Relleno de datos faltantes con la media y Random Forest

Para este primer escenario el primer paso es rellenar los datos que faltan en las demás columnas, con la media, lo cual se hizo de la siguiente manera:

```
[ ]    from sklearn.impute import SimpleImputer

       imp_median = SimpleImputer(missing_values=np.nan, strategy='median')

[ ]    df_clean = imp_median.fit_transform(newData.iloc[:, 1:-1])

[ ]    df_impute = pd.DataFrame(data = df_clean, columns = newData.columns[1:-1])
```

Donde se toma el dataset al cual se le borraron las columnas anteriormente y se le aplica una técnica de relleno de datos, sin tener en cuenta la primera y la última columna, las cuáles representan un TAG o id de los datos y la variable objetivo respectivamente, y obtenemos un nuevo dataset solo con las características y con los datos rellenados con la media

Ya con los datos limpios, se procede a separar los datos en X, y Y, donde X almacena las características y Y la variable objetivo a predecir.

```
[ ]    X = df_impute
       Y = newData['emission']
```

Y se procede a dividir los datos en dos conjuntos, uno para entrenamiento y otro para evaluar el rendimiento del modelo, lo cual se hace de la siguiente manera

```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_state=45)

[ ]    clf = RandomForestRegressor(random_state = 45, n_jobs=-1)

[ ]    clf.fit(X_train, y_train)

       RandomForestRegressor
       RandomForestRegressor(n_jobs=-1, random_state=45)
```

Donde se tiene un test_size de 0.3, lo que significa que el 30% de los datos se asignan al conjunto de prueba, y el 70% restante se usarán para entrenamiento.

Ya con los conjuntos de datos obtenidos, se procede a entrenar los datos con el modelo de Random Forest y se obtuvieron los siguientes resultados

```
[ ]    y_pred = clf.predict(X_test)

[ ]    print(f'RMSE Score: {mean_squared_error(y_test, y_pred, squared=False)}')
       print("R^2 : ", r2_score(y_test, y_pred))
       print("MAE :", mean_absolute_error(y_test,y_pred))

    RMSE Score: 22.535370367758908
    R^2 :  0.9747592356050864
    MAE : 8.773575390916621
```

Donde con esta decisión de limpieza, se obtuvieron los siguientes resultados:

RMS de aproximadamente 22.54, lo que significa que en promedio las predicciones del modelo tiene un error alrededor de 22.54 unidades. Esta métrica nos indica cuán cerca están las predicciones del modelo de los valores reales.
R^2 de aproximadamente 0.975, lo que significa que el modelo es capaz de explicar aproximadamente el 97.5% de la variabilidad de los datos, lo cual en este caso es muy bueno indicador, ya que entre más cercano a 1 significa que el modelo explica una gran cantidad de variabilidad en los datos.
MAE de aproximadamente 8.77, lo cual significa que en promedio las predicciones del modelo. Esta métrica nos indica el promedio de las diferencias absolutas entre las predicciones del modelo y los valores reales.

Eliminar filas con datos faltantes y XGBC

La segunda variable de decisión de limpieza fue la eliminación de datos faltantes. Consiguiente aplicamos el modelo XGC.
En el siguiente paso creamos una versión de dataframe en la cual consta de la eliminación de filas que contiene datos vacíos, ya que había más de 20 mil filas con datos faltantes, queríamos comprobar cuál de las dos versiones tenía mejores resultados.

```
df_row_clean = newData.dropna(0)
```

Realizamos la asignación de la segunda versión del dataframe a XRC, análogo a X.

```
[ ]  X_RC = df_row_clean.drop(['ID_LAT_LON_YEAR_WEEK', 'emission'], axis = 1).fillna(0)
```

Asignación a Y_RC la variable objetivo “emission”.

```
[ ]  Y_RC = df_row_clean['emission']
```

Realizamos la división de datos en dos, uno de entrenamiento, y otro de evaluación de rendimiento, de la siguiente manera.

```
[ ] X_RC_train, X_RC_test, Y_RC_train, Y_RC_test = train_test_split(X_RC, Y_RC, test_size = 0.3, random_state = 42)

xgb = XGBRegressor()

[ ] xgb.fit(X_RC_train, Y_RC_train)
```

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=None, n_jobs=None,
             num_parallel_tree=None, random_state=None, ...)
```

Con los conjuntos de datos obtenidos, se procede a entrenar los datos con el modelo XGCB y se obtuvieron los siguientes resultados:

```
[ ] y_rc_pred = xgb.predict(X_RC_test)

print(f'RMSE Score: {mean_squared_error(Y_RC_test, y_rc_pred, squared=False)}')
print("R^2 : ", r2_score(Y_RC_test, y_rc_pred))
print("MAE :", mean_absolute_error(Y_RC_test,y_rc_pred))
```

```
RMSE Score: 25.709075973684286
R^2 :  0.9657398312435168
MAE : 13.879364612260398
```

RMS de aproximadamente 25.71, lo que significa que en promedio las predicciones del modelo tiene un error alrededor de 25.71 unidades. Esta métrica nos indica cuán cerca están las predicciones del modelo de los valores reales.

R^2 de aproximadamente 0.966, lo que significa que el modelo es capaz de explicar aproximadamente el 96.6% de la variabilidad de los datos, lo cual en este caso es muy bueno indicador, ya que entre más cercano a 1 significa que el modelo explica una gran cantidad de variabilidad en los datos.

MAE de aproximadamente 13.88, lo cual significa que en promedio las predicciones del modelo. Esta métrica nos indica el promedio de las diferencias absolutas entre las predicciones del modelo y los valores reales.