DengAI: Predicción de la propagación de enfermedades

Sumario

1. Introducción

La presente actividad tiene como objetivo abordar el problema de predicción de casos de dengue en dos ciudades (San Juan y Iquitos), utilizando técnicas de aprendizaje automático. Esta tarea forma parte de la competición **"DengAI: Predicting Disease Spread"** organizada por DrivenData.

2. Dataset

El conjunto de datos proporcionado incluye variables meteorológicas y ambientales semanales, además del número total de casos registrados. Se utilizaron dos archivos principales:

- `dengue_features_train.csv`
- `dengue_labels_train.csv`

Estos archivos fueron combinados y ordenados cronológicamente por ciudad y fecha de inicio de semana.

3. Preprocesamiento

Se realizaron los siguientes pasos:

- Imputación de valores faltantes (mediana).

```
X_train = X_train.fillna(X_train.median())
X_test = X_test.fillna(X_test.median())
```

- Escalado de características numéricas con `StandardScaler`.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- Codificación One-Hot para la variable categórica `city`.

```
data = pd.get_dummies(data, columns=['city'])
```

- Extracción de características temporales: mes, día y día de la semana.

```
data['month'] = data['week_start_date'].dt.month
data['day'] = data['week_start_date'].dt.day
data['day_of_week'] = data['week_start_date'].dt.dayofweek
```

- Aplicación de `log1p()` a la variable objetivo (`total_cases`) para suavizar su distribución.

```
y_train = np.log1p(y_train)
```

4. Función de entrenamiento

La función “train_predict_evaluate” es la que se encarga de realizar el entrenamiento de los modelos. Aquí muestro su funcionamiento:

Primero, elige qué modelo vamos a usar (`RandomForest` o `KNN`) y con qué método queremos ajustar los parámetros: puede ser `GridSearch`, `RandomSearch` o simplemente entrenarlo normal (`None`).

- Si usamos `GridSearch`, prueba todas las combinaciones posibles de parámetros.
- Si usamos `RandomSearch`, prueba combinaciones al azar (pero es más rápido).
- Si elegimos `None`, entrena el modelo tal cual viene de fábrica, sin buscar mejores parámetros.

Después, entrena el modelo con los datos de entrenamiento (`X_train`, `y_train`).

Cuando ya está entrenado, hace predicciones sobre los datos de prueba (`X_test`).
Ojo: como los datos están en escala logarítmica (por el `log1p()`), aquí se revierte esa transformación con `expm1()` para que los números vuelvan a su escala normal.

Luego calcula el **MAE (Mean Absolute Error)**, que nos dice en promedio cuánto se está equivocando el modelo. También imprime algunas estadísticas (como media y desviación estándar) de los valores reales, y calcula qué porcentaje representa el error respecto a la media.

Al final, muestra todo eso en consola para que podamos ver qué tan bien o mal fue el modelo.

Esta función es demasiado larga como para enseñarla en una captura. Sugiero verla en github directamente.

4. Modelos utilizados

Se aplicaron dos algoritmos distintos:

1. **K-Nearest Neighbors (KNN)**
2. **Random Forest Regressor**

3. **Naive Bayes** *(No es posible aplicarlo, ya que es un modelo de clasificación, y esto es un problema de regresión)*

Cada modelo fue entrenado y evaluado utilizando dos técnicas de búsqueda de hiperparámetros:

- **Grid Search**
- **Random Search**

5. Evaluación

La métrica utilizada para comparar los modelos fue el **Error Absoluto Medio (MAE)** en la escala original de casos (revirtiendo la transformación logarítmica). Se imprimieron estadísticas adicionales del conjunto de prueba para contextualizar los resultados.

Se debe tener en cuenta que la media de casos es de 25.4829

Resultados observados:

Modelo	Búsqueda	MAE (original)	Mejores parámetros (si aplica)
RandomForest	GridSearch	*11.6199*	*'max_depth': 20 'max_features': 0.5 'min_samples_leaf': 3 'n_estimators': 100*
RandomForest	RandomSearch	*11.1747*	*'max_depth': 10 'max_features': 0.7, 'min_samples_leaf': 2, 'n_estimators': 108*
KNN	GridSearch	*15.3082*	*'metric': 'manhattan', 'n_neighbors': 10, 'weights': 'uniform'*
KNN	RandomSearch	*15.8116*	*'metric': 'manhattan', 'n_neighbors': 8, 'weights': 'distance'*

Este es el código principal utilizado para entrenar los modelos. Para ver el código completo, consultar en el [github](#) :

```python
    preprocessor = create_preprocessor(merged_df.drop(columns=['total_cases', 'week_start_date']))
    print("Preprocessor created.") # Mensaje de confirmación

    # 2. Preprocesar y dividir los datos, pasando el preprocesador
    try:
        # Ahora pasamos 'preprocessor' como segundo argumento
        X_train, X_test, y_train, y_test = preprocess_and_split(merged_df, preprocessor)

        if X_train is not None: # Verificar que el preprocesamiento fue exitoso
            print(f"X_train shape: {X_train.shape}") # Imprimir shapes para verificar

            # 3. Entrenar, predecir y evaluar diferentes modelos y métodos de búsqueda

            train_predict_evaluate('RandomForest', 'GridSearch', X_train, X_test, y_train, y_test)
            train_predict_evaluate('RandomForest', 'RandomSearch', X_train, X_test, y_train, y_test)
            train_predict_evaluate('KNN', 'GridSearch', X_train, X_test, y_train, y_test)
            train_predict_evaluate('KNN', 'RandomSearch', X_train, X_test, y_train, y_test)

        else:
            print("Error durante el preprocesamiento y división de datos.")

    except Exception as e:
        print(f"ERROR during preprocess_and_split call or subsequent steps: {e}")
        import traceback
        traceback.print_exc()

else:
    print("Proceso detenido debido a error en la carga de archivos.")
```

```
Preprocessor created.
--- Inside preprocess_and_split ---
Value of target_col at entry: total_cases
Type of target_col at entry: <class 'str'>
Attempting to access df column using target_col: 'total_cases'
Fitting preprocessor...
Transforming train data...
Transforming test data...
--- Exiting preprocess_and_split ---
X_train shape: (1164, 27)

--- Entrenando RandomForest usando GridSearch ---
Configurando GridSearchCV para RandomForest...
Entrenando...
Fitting 3 folds for each of 36 candidates, totalling 108 fits
c:\Users\Javier\AppData\Local\Programs\Python\Python312\Lib\site-packages\numpy\ma\core.py:2892: RuntimeWarning: invalid value encountered in cast
  _data = np.array(data, dtype=dtype, copy=copy,
Mejores parámetros encontrados:
{'max_depth': 20, 'max_features': 0.5, 'min_samples_leaf': 3, 'n_estimators': 100}
Realizando predicciones...

Mean Absolute Error (MAE) - RandomForest (GridSearch): 11.6199

Estadísticas de 'total_cases' en el conjunto de prueba (escala original):
- Media:      25.4829
- Mediana:    13.0000
- Desv. Est.: 41.7328
- Mínimo:     0.0000
- Máximo:     410.0000
```

6. Conclusiones

- El modelo **Random Forest** mostró mejor desempeño general, especialmente cuando se aplicó búsqueda de hiperparámetros.
- **KNN** requiere un preprocesamiento cuidadoso (escalado) y fue sensible al número de vecinos.
- Se observó una mejora notable al aplicar técnicas de optimización de hiperparámetros.
- La transformación `log1p` ayudó a reducir la varianza en los errores.

7. Enlaces de referencia

- **Enlace a github : [github]**