perplexity

Análisis del conjunto de datos "Individual Household Electric Power Consumption"

El conjunto de datos "Individual Household Electric Power Consumption" contiene mediciones del consumo eléctrico de un hogar ubicado en Sceaux (a 7 km de París, Francia) durante casi 4 años, desde diciembre de 2006 hasta noviembre de 2010[1]. Este conjunto de datos es multivariate y de series temporales, con un muestreo por minuto que resulta en más de 2 millones de observaciones[2].

Descripción del conjunto de datos

El conjunto de datos contiene las siguientes variables[1] [2]:

- **Date**: Fecha en formato dd/mm/yyyy
- **Time**: Hora en formato hh:mm:ss
- **Global_active_power**: Potencia activa global consumida (en kilovatios)
- **Global_reactive_power**: Potencia reactiva global consumida (en kilovatios)
- **Voltage**: Voltaje promedio (en voltios)
- **Global_intensity**: Intensidad de corriente promedio (en amperios)
- **Sub_metering_1**: Energía activa para la cocina (vatios-hora)
- **Sub_metering_2**: Energía activa para la lavandería (vatios-hora)
- **Sub_metering_3**: Energía activa para sistemas de climatización (vatios-hora)

Es importante destacar que el conjunto de datos contiene algunos valores faltantes (aproximadamente 1,25% de las filas)[1].

Preprocesamiento de los datos

Para preparar los datos para su análisis, debemos realizar los siguientes pasos:

1. **Carga de datos**: Cargar el archivo de texto y convertir las columnas de fecha y hora en un índice datetime.
2. **Limpieza de valores faltantes**: Reemplazar los valores '?' por NaN y decidir cómo manejarlos.
3. **Creación de variables adicionales**: Calcular el sub_metering_4 que representa el consumo de energía no medido por los otros sub-meterings.
4. **Normalización**: Escalar los datos para mejorar el rendimiento de los modelos.

Código para el preprocesamiento

```
# Importar bibliotecas necesarias
import pandas as pd
import numpy as np
from numpy import nan
from matplotlib import pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error
import math

# Cargar el conjunto de datos
dataset = pd.read_csv('household_power_consumption.txt', sep=';', header=0,
                      low_memory=False, infer_datetime_format=True,
                      parse_dates={'datetime':[0,1]}, index_col=['datetime'])

# Reemplazar valores faltantes
dataset.replace('?', nan, inplace=True)

# Convertir a tipo float
values = dataset.values.astype('float32')

# Añadir columna para el resto de sub-metering
dataset['sub_metering_4'] = (values[:,0] * 1000 / 60) - (values[:,4] + values[:,5] + valu

# Guardar el dataset procesado
dataset.to_csv('household_power_consumption_clean.csv')
```

Exploración de datos

La exploración de los datos nos permite entender mejor los patrones de consumo eléctrico en este hogar. Podemos observar:

- Patrones diarios de consumo
- Variaciones semanales
- Tendencias estacionales
- Períodos de bajo consumo (posiblemente vacaciones)

Visualización de datos

```
# Cargar datos limpios
dataset = pd.read_csv('household_power_consumption_clean.csv', header=0,
                      infer_datetime_format=True, parse_dates=['datetime'],
                      index_col=['datetime'])

# Visualizar el consumo de energía activa global
plt.figure(figsize=(12, 6))
plt.plot(dataset['Global_active_power'])
plt.title('Consumo de energía activa global')
plt.xlabel('Fecha')
plt.ylabel('Potencia activa global (kilovatios)')
```

```
plt.show()

# Visualizar los diferentes sub-meterings
plt.figure(figsize=(12, 8))
plt.subplot(4, 1, 1)
plt.plot(dataset['Sub_metering_1'])
plt.title('Consumo en cocina')
plt.subplot(4, 1, 2)
plt.plot(dataset['Sub_metering_2'])
plt.title('Consumo en lavandería')
plt.subplot(4, 1, 3)
plt.plot(dataset['Sub_metering_3'])
plt.title('Consumo en climatización')
plt.subplot(4, 1, 4)
plt.plot(dataset['sub_metering_4'])
plt.title('Otros consumos')
plt.tight_layout()
plt.show()
```

Preparación para modelos de series temporales

Para preparar los datos para los modelos LSTM y GRU, necesitamos:

1. Remuestrear los datos a una frecuencia más manejable (por ejemplo, horaria)
2. Crear secuencias de entrada-salida
3. Normalizar los datos
4. Dividir en conjuntos de entrenamiento y prueba

```
# Remuestrear a frecuencia horaria
hourly_data = dataset.resample('H').mean()

# Seleccionar características para el modelo
features = ['Global_active_power', 'Global_reactive_power', 'Voltage',
            'Global_intensity', 'Sub_metering_1', 'Sub_metering_2',
            'Sub_metering_3', 'sub_metering_4']
data = hourly_data[features]

# Normalizar datos
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data)

# Función para crear secuencias de entrada-salida
def create_sequences(data, seq_length):
    X, y = [], []
    for i in range(len(data) - seq_length):
        X.append(data[i:i + seq_length, :])
        y.append(data[i + seq_length, 0])  # Predecir Global_active_power
    return np.array(X), np.array(y)

# Crear secuencias (24 horas para predecir la siguiente)
seq_length = 24
X, y = create_sequences(scaled_data, seq_length)
```

```
# Dividir en entrenamiento y prueba (80/20)
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Implementación del modelo LSTM

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Construir modelo LSTM
model_lstm = Sequential()
model_lstm.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[^1], X_train.sh
model_lstm.add(Dropout(0.2))
model_lstm.add(LSTM(50))
model_lstm.add(Dropout(0.2))
model_lstm.add(Dense(1))

# Compilar modelo
model_lstm.compile(optimizer='adam', loss='mse')

# Entrenar modelo
history_lstm = model_lstm.fit(
    X_train, y_train,
    epochs=50,
    batch_size=72,
    validation_split=0.2,
    verbose=1
)

# Evaluar modelo
y_pred_lstm = model_lstm.predict(X_test)
```

Implementación del modelo GRU

```
from tensorflow.keras.layers import GRU

# Construir modelo GRU
model_gru = Sequential()
model_gru.add(GRU(50, return_sequences=True, input_shape=(X_train.shape[^1], X_train.shap
model_gru.add(Dropout(0.2))
model_gru.add(GRU(50))
model_gru.add(Dropout(0.2))
model_gru.add(Dense(1))

# Compilar modelo
model_gru.compile(optimizer='adam', loss='mse')

# Entrenar modelo
history_gru = model_gru.fit(
    X_train, y_train,
    epochs=50,
    batch_size=72,
```

```
    validation_split=0.2,
    verbose=1
)

# Evaluar modelo
y_pred_gru = model_gru.predict(X_test)
```

Evaluación y comparación de modelos

Para evaluar los modelos, utilizaremos las métricas RMSE (Error Cuadrático Medio) y MAE (Error Absoluto Medio):

```
# Invertir la normalización para obtener valores reales
y_test_inv = scaler.inverse_transform(np.concatenate([np.zeros((len(y_test), 7)), y_test.
y_pred_lstm_inv = scaler.inverse_transform(np.concatenate([np.zeros((len(y_pred_lstm), 7)
y_pred_gru_inv = scaler.inverse_transform(np.concatenate([np.zeros((len(y_pred_gru), 7)),

# Calcular métricas
rmse_lstm = math.sqrt(mean_squared_error(y_test_inv, y_pred_lstm_inv))
mae_lstm = mean_absolute_error(y_test_inv, y_pred_lstm_inv)

rmse_gru = math.sqrt(mean_squared_error(y_test_inv, y_pred_gru_inv))
mae_gru = mean_absolute_error(y_test_inv, y_pred_gru_inv)

# Mostrar resultados
print('LSTM - RMSE: %.3f, MAE: %.3f' % (rmse_lstm, mae_lstm))
print('GRU - RMSE: %.3f, MAE: %.3f' % (rmse_gru, mae_gru))

# Visualizar predicciones
plt.figure(figsize=(12, 6))
plt.plot(y_test_inv[:100], label='Real')
plt.plot(y_pred_lstm_inv[:100], label='LSTM')
plt.plot(y_pred_gru_inv[:100], label='GRU')
plt.title('Predicción de consumo eléctrico')
plt.xlabel('Horas')
plt.ylabel('Global_active_power (kW)')
plt.legend()
plt.show()

# Tabla comparativa
comparison = pd.DataFrame({
    'Modelo': ['LSTM', 'GRU'],
    'RMSE': [rmse_lstm, rmse_gru],
    'MAE': [mae_lstm, mae_gru]
})
print(comparison)
```

Análisis de resultados

Al comparar los modelos LSTM y GRU para la predicción del consumo eléctrico, podemos observar:

Fortalezas y debilidades del LSTM:

- **Fortalezas**: Capacidad para capturar dependencias a largo plazo en los datos de consumo eléctrico.
- **Debilidades**: Mayor complejidad computacional y tiempo de entrenamiento más largo.

Fortalezas y debilidades del GRU:

- **Fortalezas**: Arquitectura más simple con menos parámetros, lo que resulta en un entrenamiento más rápido.
- **Debilidades**: Puede tener dificultades para capturar patrones muy complejos a largo plazo.

En este caso específico, ambos modelos pueden mostrar rendimientos similares debido a que los patrones de consumo eléctrico suelen tener una estructura temporal bien definida (patrones diarios, semanales y estacionales).

Conclusiones

El conjunto de datos "Individual Household Electric Power Consumption" proporciona una excelente oportunidad para aplicar modelos de series temporales como LSTM y GRU para predecir el consumo eléctrico. Ambos modelos son capaces de capturar los patrones temporales en los datos, aunque pueden diferir en términos de eficiencia computacional y precisión.

Para mejorar aún más los resultados, se podrían considerar:

- Incorporar variables externas como temperatura o días festivos
- Experimentar con diferentes arquitecturas de red
- Utilizar técnicas de ensemble combinando múltiples modelos

Este análisis demuestra la utilidad de las redes neuronales recurrentes para la predicción del consumo eléctrico, lo que puede ayudar a los hogares y empresas a optimizar su uso de energía y reducir costos.

⁂

1. https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption
2. https://archive.ics.uci.edu/dataset/235/individual+household+electric+power+consumption