Documentación Modelo de predicción de Temperatura

Valentina Polania Osorio– 20201188877

Facultad de Ingeniería, Universidad Surcolombiana

Inteligencia Artificial

Juan Castro

30 mayo 2025

Modelo de Predicción de Temperatura

1. Arquitectura General

El flujo general de desarrollo es el siguiente:

generador_datos.py → temperaturas_simuladas.csv → entrenamiento_modelo.py → modelo_temperatura.pkl → predictor.py → probar_predictor.py

2. Generación de Datos – generador_datos.py

Genera un conjunto de datos sintéticos que simulan lecturas de temperatura minuto a minuto durante 7 días consecutivos, a partir del 1 de mayo de 2025.

```
5   rng = np.random.default_rng(seed=42)
6
7   minutos_totales = 7 * 24 * 60
8
9   temperaturas = []
10  timestamps = []
11
12  inicio = datetime(2025, 5, 1, 0, 0, 0)
```

El modelo de simulación se basa en un comportamiento térmico diario, replicado mediante una función sinusoidal que imita los cambios de temperatura típicos de un ciclo de 24 horas: temperaturas mínimas alrededor de las 3:00 a.m. (~23 °C) y máximas cerca del mediodía (~38 °C). Para agregar realismo, se incluye ruido aleatorio con una distribución normal de media 0 y desviación estándar 0.5 °C.

```
15  for minuto in range(minutos_totales):
16      current_time = inicio + timedelta(minutes=minuto)
17      timestamps.append(current_time)
18
19      hora_decimal = current_time.hour + current_time.minute / 60.0
20
21      temp_base = 30.5 + 7.5 * np.sin((hora_decimal - 6) * np.pi / 12)
22
23      ruido = rng.normal(loc=0, scale=0.5)
24      temperatura = round(temp_base + ruido, 2)
25      temperaturas.append(temperatura)
26
```

Cada lectura es asociada con un timestamp exacto, y el resultado se guarda en un archivo temperaturas_simuladas.csv con dos columnas:

```python
df = pd.DataFrame({
    "timestamp": timestamps,
    "temperatura": temperaturas
})

df.to_csv("temperaturas_simuladas.csv", index=False)
```

- timestamp: fecha y hora correspondiente a cada lectura.
- temperatura: valor simulado en grados Celsius.

3. Entrenamiento del Modelo – entrenamiento_modelo.py

Se entrena un modelo de aprendizaje supervisado para predecir la temperatura dentro de una hora, usando como entrada los 60 minutos anteriores. El proceso parte del archivo temperaturas_simuladas.csv generado previamente.

1. Lectura de Datos:

```python
df = pd.read_csv("temperaturas_simuladas.csv")

temperaturas = df["temperatura"].values

input_size = 60     # Últimos 60 minutos como entrada
pred_horizon = 60   # Predicción 60 minutos en el futuro

# Preparar ventanas de entrenamiento
X = []
y = []
```

- Se carga el archivo CSV con las temperaturas simuladas.
- Se extrae la columna 'temperatura' como array.

2. Configuración de Parámetros:

- input_size = 60: el modelo toma los últimos 60 minutos como entrada.
- pred_horizon = 60: predice la temperatura que ocurrirá una hora después.

3. Preparación del Dataset:

```python
for i in range(len(temperaturas) - input_size - pred_horizon):
    entrada = temperaturas[i : i + input_size]
    salida = temperaturas[i + input_size + pred_horizon - 1]
    X.append(entrada)
    y.append(salida)

X = np.array(X)
y = np.array(y)
```

- Se generan ventanas deslizantes con entradas X (60 temperaturas) y salidas y (temperatura futura).

4. División de Datos:

```
31  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Se usa train_test_split para dividir 80% entrenamiento y 20% prueba.

5. Entrenamiento del Modelo:

```
33  modelo = LinearRegression()
34  modelo.fit(X_train, y_train)
```

- Se entrena un modelo LinearRegression con los datos de entrenamiento.

6. Evaluación del Modelo:

```
37  y_pred = modelo.predict(X_test)
38  mae = mean_absolute_error(y_test, y_pred)
39  rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

- Se calcula el MAE y RMSE sobre el conjunto de prueba.

7. Guardado del Modelo:

```
46  joblib.dump(modelo, "modelo_temperatura.pkl")
```

- Se guarda el modelo entrenado en 'modelo_temperatura.pkl' usando joblib.

5. Predicción – predictor.py
Archivo: predictor.py

Carga: modelo_temperatura.pkl usando joblib.

Función: predecir_temperatura(ultimas_60_temp).

Validación: la lista debe contener exactamente 60 elementos.

Salida: predicción redondeada a dos decimales.

```python
import joblib    Import "joblib" could not be resolved
import numpy as np    Import "numpy" could not be resolved

modelo = joblib.load("modelo_temperatura.pkl")

def predecir_temperatura(ultimas_60_temp):
    """
    Recibe una lista o array con 60 temperaturas recientes (últimos 60 minutos),
    y devuelve la predicción de temperatura para dentro de 1 hora.
    """
    if len(ultimas_60_temp) != 60:
        raise ValueError("La lista debe contener exactamente 60 valores de temperatura.")

    entrada = np.array(ultimas_60_temp).reshape(1, -1)
    prediccion = modelo.predict(entrada)[0]
    return round(prediccion, 2)
```

6. Prueba del Modelo – probar_predictor.py
Archivo: probar_predictor.py

Simulación: Lista de 60 temperaturas con pequeñas variaciones.

Uso: Llama a la función predecir_temperatura.

Salida esperada: Predicción exitosa mostrada en consola.

```python
from predictor import predecir_temperatura

# Simulamos una lista de 60 temperaturas recientes (ejemplo estático)
ultimas_60 = [30.5 + 0.2 * (i % 5) for i in range(60)]  # pequeñas variaciones realistas

try:
    prediccion = predecir_temperatura(ultimas_60)
    print(f" Predicción generada exitosamente: {prediccion} °C")
except Exception as e:
    print(f" Error al predecir: {e}")
```

7. Requisitos del Entorno
Python >= 3.8

joblib==1.5.0

numpy==1.24.4

pandas==1.5.3

scikit-learn==1.2.2

8. Implementación en Producción
Archivos para producción: modelo_temperatura.pkl, predictor.py, requirements.txt.

Opciones de despliegue: Docker o AWS Lambda.

Entrada esperada: JSON con lista de 60 valores.

Salida: JSON con predicción de temperatura.

9. Limitaciones

Modelo lineal: no captura relaciones no lineales complejas.

No considera variables adicionales (humedad, presión).

Entrenado con datos simulados.