Explicación Completa del Workflow de ML para Análisis de Cargas en Turbinas Eólicas

Documentación Técnica

7 de febrero de 2026

Resumen

Este documento proporciona una explicación exhaustiva del script `complete_workflow.py`, un sistema interactivo de procesamiento de datos y feature engineering para el análisis de cargas en turbinas eólicas mediante técnicas de Machine Learning. El workflow procesa simulaciones de DNV Bladed, crea features avanzadas en el dominio de la frecuencia (transformaciones Coleman), y prepara datos para modelos predictivos de cargas individuales de pala (LIPC).

Índice

1. Introducción y Contexto

1.1. Propósito del Script

El script `complete_workflow.py` es un **workflow completo y ejecutable** generado automáticamente desde un notebook de Jupyter. Su objetivo principal es:

- Procesar datos de simulaciones aeroelásticas de turbinas eólicas (DNV Bladed)
- Extraer señales físicas relevantes (cargas, vientos, ángulos, etc.)
- Crear features avanzadas para Machine Learning tradicional
- Preparar datasets para modelos predictivos de cargas en palas
- Analizar la estructura espacial del viento medida por LIDAR

1.2. Arquitectura del Sistema

El sistema está diseñado con una arquitectura modular e interactiva:

1. **Ejecución interactiva:** El usuario puede ejecutar secciones individuales
2. **Sanitización de código:** Elimina comandos mágicos de Jupyter (`%`, `!`)
3. **Detección de visualizaciones:** Identifica y pregunta antes de ejecutar plots
4. **Manejo de errores:** Continúa la ejecución o aborta según decisión del usuario

2. STEP 1: Configuración Inicial del Entorno

2.1. Librerías de Python

El workflow importa y configura las siguientes categorías de librerías:

2.1.1. Librerías Estándar

- `os`, `sys`, `pathlib`: Manejo del sistema de archivos
- `warnings`: Control de advertencias
- `re`: Expresiones regulares para parsing de nombres

2.1.2. Análisis de Datos

- `pandas`: Manipulación de dataframes y series temporales
- `numpy`: Operaciones numéricas vectorizadas
- `joblib`: Serialización de modelos y datos

2.1.3. Visualización

- `matplotlib.pyplot`: Gráficos base
- `seaborn`: Visualizaciones estadísticas avanzadas
- Estilo configurado: `seaborn-v0_8-darkgrid`

2.1.4. Machine Learning

- `scikit-learn`: Modelos (Ridge, RandomForest, GradientBoosting, etc.)
- `sklearn.preprocessing`: Escaladores (StandardScaler, MinMaxScaler)
- `sklearn.metrics`: Métricas de evaluación (MSE, R^2, MAE)
- `xgboost` (opcional): Extreme Gradient Boosting
- `lightgbm` (opcional): Light Gradient Boosting Machine

2.1.5. Procesamiento de Señales

- `scipy.signal`: Filtros digitales (Butterworth)
- `scipy.interpolate`: Interpolación para cálculos de intensidad de turbulencia

2.2. Estructura de Directorios

El workflow establece la siguiente jerarquía de carpetas:

Directorio	Propósito
`data_train`	Datos de entrenamiento originales (CSVs de Bladed)
`data_train_traditional_ML`	Datos procesados con features de ML
`_report`	Reportes y análisis exploratorios
`notebook/03_ML_traditional_models`	Modelos entrenados guardados
`_results`	Resultados de predicciones
`notebook/01_Models_scaler`	Escaladores guardados
`notebook/03_Models_training`	Artefactos de entrenamiento

Cuadro 1: Estructura de directorios del proyecto

3. STEP 2: Importación desde Simulaciones Bladed

3.1. Librería PostProcessBladed

El workflow integra `postprocessbladed`, una librería personalizada para leer archivos binarios de DNV Bladed (formato `.$TE`).

3.2. Señales Extraídas

El sistema puede extraer múltiples categorías de señales:

3.2.1. Variables Aerodinámicas (Aero_B1, Aero_B2)

Velocidades del viento incidente en diferentes posiciones radiales de las palas:

- Blade 1/2 Incident axial wind speed at 0m, 6m, 18m, 30m, 46m, 59m, 68.25m

3.2.2. Cargas en el Hub Rotante

Variables objetivo (targets) para predicción:

- `Blade root 1 My`: Momento flector out-of-plane pala 1 [kNm]
- `Blade root 2 My`: Momento flector out-of-plane pala 2 [kNm]

3.2.3. Sistema de Control

- `Blade 1/2 pitch angle`: Ángulos de pitch [deg]
- `Blade 1/2 pitch rate`: Velocidades angulares de pitch [deg/s]

3.2.4. Drivetrain y Rotor

- `Rotor azimuth angle`: Ángulo azimutal del rotor [deg]
- `Rotor speed`: Velocidad de rotación [rpm]

3.2.5. Mediciones LIDAR (External_controller)

Variables de velocidad de línea de visión (VLOS) del LIDAR:

- `LAC_VLOS_BEAM0_RANGE5` a `LAC_VLOS_BEAM7_RANGE5`
- `OPER_MEAS_YAWERROR`: Error de alineación yaw [deg]

3.3. Procesamiento de Archivos

El workflow:

1. Busca archivos `.$TE` en la carpeta de simulaciones
2. Lee headers con `pp.read_hdr_files()`
3. Lee datos binarios con `pp.read_bin_files()`
4. Maneja señales Aero con posiciones radiales específicas
5. Genera un CSV por cada archivo de simulación procesado

4. STEP 3: Feature Engineering para ML Tradicional

Esta es la sección más compleja y crítica del workflow. Aquí se crean features avanzadas que capturan la física del sistema.

4.1. Lags Temporales de VLOS

4.1.1. Motivación Física

El viento tarda un tiempo en propagarse desde la posición de medición LIDAR hasta las palas del rotor. Los lags capturan esta dinámica de advección.

4.1.2. Implementación

La función `create_vlos_lags()` crea versiones desplazadas temporalmente:

```
def create_vlos_lags(df, lag_seconds_list=[2,5,8,11,14,17,20,23,26]):
    # Detecta columnas LAC_VLOS
    # Calcula dt desde columna Time
    # Para cada lag en segundos:
    #   lag_samples = int(round(lag_sec / dt))
    #   df[f"{col}_lag{lag_sec}s"] = df[col].shift(lag_samples)
```

Rango de lags: 2-26 segundos (configurable)
Output: Si hay 8 beams y 9 lags → 72 nuevas columnas

4.2. Componentes Trigonométricas de Ángulos

4.2.1. Rotor Azimuth Angle

Para evitar discontinuidades en 0/360, se crean componentes cíclicas:

$$\sin(\psi) = \sin(\text{azimuth}) \tag{1}$$
$$\cos(\psi) = \cos(\text{azimuth}) \tag{2}$$

Donde ψ es el ángulo azimutal del rotor.

4.2.2. Yaw Error

Similar transformación para el error de yaw:

$$\sin(\theta_{yaw}) = \sin(\text{OPER\_MEAS\_YAWERROR}) \tag{3}$$
$$\cos(\theta_{yaw}) = \cos(\text{OPER\_MEAS\_YAWERROR}) \tag{4}$$

Beneficio: El modelo puede aprender patrones cíclicos sin problemas de frontera.

4.3. Transformación Coleman y Componentes 0P, 1P, 2P

4.3.1. Fundamento Teórico

La transformación Coleman (también conocida como Multi-Blade Coordinate, MBC) transforma variables del marco rotatorio al marco fijo (estacionario). Esto es crucial para separar componentes de frecuencia.

4.3.2. Señales Suma y Diferencia

A partir de los momentos flectores $M_1(t)$ y $M_2(t)$:

$$M_\Sigma(t) = \frac{M_1(t) + M_2(t)}{2} \quad \text{(contiene componentes pares: 2P, 4P, ...)} \tag{5}$$
$$M_\Delta(t) = \frac{M_1(t) - M_2(t)}{2} \quad \text{(contiene componentes impares: 1P, 3P, ...)} \tag{6}$$

4.3.3. Componente 0P (Colectivo/Lento)

El componente 0P representa la carga promedio o colectiva:

$$M_0(t) = \text{LPF}(M_\Sigma(t), f_{cutoff} = 0{,}5 \cdot f_{1P}) \tag{7}$$

Donde LPF es un filtro pasa-bajo Butterworth que elimina componentes 2P, 4P, etc.

4.3.4. Componente 1P (Once Per Revolution)

El componente 1P se proyecta en ejes fijos mediante:

$$M_{1c}(t) = M_{\Delta,filtered}(t) \cdot \cos(\psi(t)) \quad \text{(componente en fase)} \tag{8}$$
$$M_{1s}(t) = M_{\Delta,filtered}(t) \cdot \sin(\psi(t)) \quad \text{(componente en cuadratura)} \tag{9}$$

Donde $M_{\Delta,filtered}$ es la señal diferencial filtrada con pasa-banda alrededor de f_{1P}.

4.3.5. Componente 2P (Twice Per Revolution)

El componente 2P se proyecta usando el doble del ángulo azimutal:

$$M_{2c}(t) = M_{\Sigma,filtered}(t) \cdot \cos(2\psi(t)) \quad \text{(componente en fase)} \tag{10}$$
$$M_{2s}(t) = M_{\Sigma,filtered}(t) \cdot \sin(2\psi(t)) \quad \text{(componente en cuadratura)} \tag{11}$$

4.3.6. Filtrado Digital

El workflow implementa filtros Butterworth de segundo orden:

```python
def bandpass_filter(signal, lowcut, highcut, fs, order=2):
    from scipy.signal import butter, sosfilt
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    sos = butter(order, [low, high], btype='band', output='sos')
    filtered = sosfilt(sos, signal)
    return filtered
```

Parámetros típicos:

- Banda 1P: $f_{1P} \pm 0{,}3$ Hz
- Banda 2P: $f_{2P} \pm 0{,}5$ Hz

4.3.7. Vector de Targets

El vector de salida resultante es:

$$\mathbf{y}(t) = [M_0(t), M_{1c}(t), M_{1s}(t), M_{2c}(t), M_{2s}(t)]^T \tag{12}$$

Este vector captura completamente la dinámica de las cargas en el marco fijo.

4.4. Features Coleman de Pitch

Similar a los momentos, los ángulos de pitch se transforman:

4.4.1. Componentes Colectivo y Diferencial

$$\theta_0(t) = \frac{\theta_1(t) + \theta_2(t)}{2} \quad \text{(pitch colectivo)} \tag{13}$$
$$\theta_\Delta(t) = \frac{\theta_1(t) - \theta_2(t)}{2} \quad \text{(pitch diferencial)} \tag{14}$$

4.4.2. Proyección 1P del Diferencial

$$\theta_{1c}(t) = \theta_\Delta(t) \cdot \cos(\psi(t)) \tag{15}$$
$$\theta_{1s}(t) = \theta_\Delta(t) \cdot \sin(\psi(t)) \tag{16}$$

4.4.3. Rates (Derivadas Temporales)

$$\dot{\theta}_0(t) \approx \frac{\theta_0(t) - \theta_0(t - \Delta t)}{\Delta t} \tag{17}$$

$$\dot{\theta}_{1c}(t) \approx \frac{\theta_{1c}(t) - \theta_{1c}(t - \Delta t)}{\Delta t} \tag{18}$$

$$\dot{\theta}_{1s}(t) \approx \frac{\theta_{1s}(t) - \theta_{1s}(t - \Delta t)}{\Delta t} \tag{19}$$

$$\dot{\Omega}(t) \approx \frac{\Omega(t) - \Omega(t - \Delta t)}{\Delta t} \tag{20}$$

Output: 7 nuevas columnas que capturan la dinámica del control de pitch.

4.5. Estadísticas del Campo de Viento LIDAR

4.5.1. Configuración Espacial de Beams

El LIDAR escanea el campo de viento con 8 beams distribuidos circularmente:

Beam	Ángulo
0	0° (arriba)
1	45°
2	90° (derecha)
3	135°
4	180° (abajo)
5	225°
6	270° (izquierda)
7	315°

Cuadro 2: Configuración angular de los beams LIDAR

4.5.2. Variables Estadísticas Creadas

Velocidad Media del Campo

$$U_{mean}(t) = \frac{1}{N} \sum_{i=1}^{N} \text{VLOS}_i(t) \tag{21}$$

Captura la intensidad general del viento.

Heterogeneidad/Turbulencia

$$U_{std}(t) = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (\text{VLOS}_i(t) - U_{mean}(t))^2} \tag{22}$$

Mide la variabilidad espacial del viento.

Shear Vertical

$$U_{shear,vert}(t) = \frac{1}{N_{up}} \sum_{i \in \text{beams\_up}} \text{VLOS}_i(t) - \frac{1}{N_{down}} \sum_{j \in \text{beams\_down}} \text{VLOS}_j(t) \tag{23}$$

Captura el gradiente vertical del viento (importante para 1P).

Shear Horizontal

$$U_{shear,horiz}(t) = \frac{1}{N_{left}} \sum_{i \in \text{beams\_left}} \text{VLOS}_i(t) - \frac{1}{N_{right}} \sum_{j \in \text{beams\_right}} \text{VLOS}_j(t) \tag{24}$$

Captura gradientes laterales relacionados con yaw misalignment.

4.5.3. Rotación Configurable

El workflow permite rotar la configuración de beams mediante un parámetro `rotation_offset`:

```
def rotate_beam(beam_num, offset):
    return (beam_num + offset) % 8
```

Esto es útil para alinear el sistema de coordenadas del LIDAR con el del rotor.

4.6. Lags de Estadísticas de Viento

Las cuatro variables estadísticas (U_{mean}, U_{std}, $U_{shear,vert}$, $U_{shear,horiz}$) también se desplazan temporalmente:

Output: $4 \times 9 = 36$ nuevas columnas (asumiendo 9 lags).

4.7. Intensidad de Turbulencia (TI)

4.7.1. Definición

$$\text{TI}(t) = \frac{U_{std}(t)}{U_{mean}(t)} \tag{25}$$

La intensidad de turbulencia es una métrica adimensional que caracteriza la turbulencia normalizada.

4.7.2. Condiciones Extremas

El workflow clasifica instantes como extremos si:

$$\text{TI}(t) \geq \text{TI}_{P95}(U_{mean}(t)) \tag{26}$$

Donde TI_{P95} se interpola desde archivos de referencia específicos del DLC:

- `Umean_DLC12a.txt`
- `TI_P95_DLC12a.txt`

Output: Columna binaria `Extreme_condition` (0/1).

4.8. Derivadas del Yaw

4.8.1. Yaw Rate

$$\dot{\theta}_{yaw}(t) = \frac{d(\text{OPER\_MEAS\_YAWERROR})}{dt} \tag{27}$$

Calculada con `np.gradient()` usando el paso temporal dinámico.

4.8.2. Yaw Acceleration

$$\ddot{\theta}_{yaw}(t) = \frac{d\dot{\theta}_{yaw}}{dt} \tag{28}$$

Output: 2 nuevas columnas que capturan la dinámica del control de yaw.

4.9. Cortadura Física del Viento

4.9.1. Modelo de Plano Inclinado

El sistema ajusta un modelo lineal del campo de viento en el plano del rotor:

$$U(y, z) = U_0 + \frac{\partial U}{\partial y} y + \frac{\partial U}{\partial z} z \tag{29}$$

4.9.2. Coordenadas de los Beams

Para un range a distancia D y ángulo de cono ϕ:

$$\rho = D \sin(\phi) \quad \text{(radio del anillo)} \tag{30}$$
$$y_b = \rho \sin(\theta_b) \tag{31}$$
$$z_b = \rho \cos(\theta_b) \tag{32}$$

4.9.3. Ajuste por Mínimos Cuadrados

Se proyectan las mediciones VLOS al plano perpendicular:

$$U_{proj,b} = \frac{\text{VLOS}_b}{\cos(\phi)} \tag{33}$$

Luego se resuelve el sistema:

$$\begin{bmatrix} 1 & y_0 & z_0 \\ 1 & y_1 & z_1 \\ \vdots & \vdots & \vdots \\ 1 & y_7 & z_7 \end{bmatrix} \begin{bmatrix} U_0 \\ \frac{\partial U}{\partial y} \\ \frac{\partial U}{\partial z} \end{bmatrix} = \begin{bmatrix} U_{proj,0} \\ U_{proj,1} \\ \vdots \\ U_{proj,7} \end{bmatrix} \tag{34}$$

4.9.4. Variables de Salida

Para cada range:

- `U0_RANGE{r}`: Velocidad base
- `dU_dy_RANGE{r}`: Gradiente horizontal [(m/s)/m]
- `dU_dz_RANGE{r}`: Gradiente vertical [(m/s)/m]
- `deltaU_y_diam_RANGE{r}`: $\frac{\partial U}{\partial y} \times (2\rho)$ [m/s]
- `deltaU_z_diam_RANGE{r}`: $\frac{\partial U}{\partial z} \times (2\rho)$ [m/s]

4.9.5. Lags de Shear

También se crean lags temporales de estas variables de cortadura física.

4.10. Desplazamiento Temporal de Targets

El workflow puede desplazar los targets hacia el futuro:

```
def shift_targets_forward_two_seconds(shift_seconds=2.0):
    # Para cada target:
    df[col] = df[col].shift(-shift_samples)
    # Recortar ultimas filas con NaN
```

Motivación: Predecir cargas 2 segundos en el futuro (control predictivo).

4.11. Identificador Name_DLC

Añade una columna con el nombre del archivo fuente para trazabilidad:

```
df['Name_DLC'] = '0001_DLC12a_030_000'
```

4.12. Limpieza de NaNs

Finalmente, se eliminan filas con valores faltantes generados por los lags:

```
df_cleaned = df.dropna()
```

5. Visualizaciones

El workflow incluye múltiples análisis visuales (ejecutados de forma opcional):

5.1. Análisis de Intensidad de Turbulencia

Gráficos generados:

- Scatter plot TI vs U_{mean} con bandas P5/P50/P95
- Comparación entre DLC12a y DLC13
- Exportación de arrays de percentiles a archivos .txt

5.2. Visualización de Componentes Coleman

Plots de series temporales:

- $M_0(t)$: Componente 0P
- $M_{1c}(t)$, $M_{1s}(t)$: Componentes 1P
- $M_{2c}(t)$, $M_{2s}(t)$: Componentes 2P

Reconstrucción de señales:

- Comparación Original vs Reconstrucción (0P+1P)
- Comparación Original vs Reconstrucción (0P+1P+2P)
- Análisis de error (RMSE, evolución temporal)
- Zoom en ventanas temporales específicas

5.3. Análisis de Turbulencia por Beam y Range

Heatmaps:

- TI por Range y Beam
- Media VLOS por Range y Beam

5.4. Mapas Espaciales del Campo de Viento

Scatter polar:

- Distribución circular de beams con valores coloreados
- Visualización de gradientes espaciales

5.5. Análisis de Cortadura Física

Series temporales:

- $\frac{\partial U}{\partial y}(t)$ y $\frac{\partial U}{\partial z}(t)$
- Magnitud de cortadura $||\nabla U||$
- Rolling means para tendencias

Rosa"de cortadura:

- Scatter plot $\frac{\partial U}{\partial z}$ vs $\frac{\partial U}{\partial y}$
- Histogramas de distribuciones

Mapas combinados:

- Beams LIDAR + vectores de cortadura
- Componentes horizontal y vertical del shear

6. Resumen de Features Generadas

Categoría de Feature	N° Columnas	Función
Lags VLOS (8 beams × 9 lags)	72	`create_vlos_lags`
sin/cos azimuth	2	`create_azimuth_components`
sin/cos yaw error	2	`create_yawerror_components`
Componentes 0P/1P/2P	5	`create_frequency_components_1P_2P`
Features Coleman pitch	7	`create_pitch_coleman_features`
Estadísticas viento LIDAR	4	`create_wind_field_statistics`
Lags estadísticas (4 × 9)	36	`create_wind_statistics_lags`
Intensidad turbulencia	1	`create_turbulence_intensity`
Flag condición extrema	1	`add_extreme_condition_flag`
Derivadas yaw	2	`create_yaw_derivatives`
Shear físico (5 × 1 range)	5	`compute_shear_physical_multi_range`
Lags shear (5 × 9)	45	`create_physical_shear_lags`
TOTAL APROXIMADO	**182+**	

Cuadro 3: Resumen de features creadas por el workflow

7. Consideraciones de Diseño

7.1. Frecuencia de Muestreo

El workflow detecta automáticamente la frecuencia de muestreo desde la columna `Time`:

```
dt = df['Time'].iloc[1] - df['Time'].iloc[0]
fs = 1.0 / dt  # T picamente 50 Hz (dt=0.02s)
```

7.2. Manejo de Beams Vacíos

Beams con >90 % de NaNs se descartan automáticamente:

```
nan_percentage = df[col].isna().sum() / len(df) * 100
if nan_percentage > 90:
    # Ignorar beam
```

7.3. Validación y Errores

Todas las funciones validan la presencia de columnas requeridas:

```
required_cols = ['Time', 'Rotor speed', 'Blade root 1 My', ...]
missing = [c for c in required_cols if c not in df.columns]
if missing:
    raise ValueError(f"Faltan columnas: {missing}")
```

7.4. Modularidad

Cada feature es creada por una función independiente, permitiendo:

- Reutilización en otros contextos
- Testing individual
- Modificación sin afectar otras partes
- Ejecución selectiva

8. Aplicaciones y Casos de Uso

8.1. Control Individual de Pitch (IPC)

Los features Coleman (M_1c, M_1s, M_2c, M_2s) son ideales para diseñar controladores IPC que reducen cargas asimétricas.

8.2. Control Predictivo (MPC)

El desplazamiento temporal de targets permite entrenar modelos para predicción horizonte-futuro.

8.3. Análisis de Fatiga

Las componentes de frecuencia permiten calcular ciclos de carga y estimar vida útil (DEL).

8.4. Validación de Modelos Aeroelásticos

Los análisis visuales ayudan a verificar la física capturada en las simulaciones.

8.5. Detección de Anomalías

Las estadísticas de turbulencia y flags de condiciones extremas son útiles para monitoreo.

9. Conclusiones

El script `complete_workflow.py` es un sistema completo y sofisticado que:

1. **Automatiza** el procesamiento de simulaciones Bladed
2. **Implementa** transformaciones matemáticas avanzadas (Coleman, filtrado digital)
3. **Genera** cientos de features físicamente motivadas
4. **Facilita** el análisis exploratorio mediante visualizaciones
5. **Prepara** datos listos para entrenar modelos de ML

El diseño modular y la documentación interna del código lo hacen mantenible y extensible para futuros desarrollos en control avanzado de turbinas eólicas.