PROYECTO FINAL: CLASIFICACIÓN DE ACTIVIDADES HUMANAS CON SENSORES

Brittany Ibling Marino Quispe, Yomar Rita Guzmán Morales, Rosario Calisaya Calderon

MSc. Ing. Paola Alejandra Carranza Bravo

Resumen.- En este informe se tiene a detalle el desarrollo de un sistema inteligente que reconoce actividades humanas (caminar, correr, subir/bajar escaleras, etc.) a partir de datos de sensores, utilizando modelos supervisados (Random Forest y SVM) y no supervisados (clustering jerárquico), con análisis de métricas, visualizaciones y conclusiones sobre el rendimiento de cada enfoque..

I. OBJETIVO

1. Objetivo General.

Desarrollar un modelo de aprendizaje supervisado/ No Supervisado que permita resolver un problema de clasificación o regresión, aplicando los conceptos y técnicas vistas en el módulo.

II. CARGAR Y EXPLORAR LOS DATOS.

```
df, actividades, features = crear_dataset_actividades_1000()

print("Primeras 5 filas del DataFrame:")
print(df.head())

print("\nInformación general del DataFrame:")
df.info()
print("\nEstadísticas descriptivas del DataFrame:")
print(df.describe())
```

```
Primeras 5 filas del DataFrame:
   acc_x_mean  acc_x_std  acc_x_max  acc_y_mean  acc_y_std  acc_y_max  \
0    0.095410   0.031120   0.161113    0.125459   0.015548   0.150835
1    0.894009   0.448095   1.612838    0.708957   0.287253   1.017876
2    0.187754   0.030248   0.356542    0.196023   0.042179   0.263424
3    1.299387   0.752584   1.714107    0.904605   0.879109   1.666050
4    1.024572   0.493215   0.848824    0.630699   0.313313   1.467082

   acc_z_mean  acc_z_std  acc_z_max  acc_magnitude  ...  gyro_z_std  \
0    0.121850   0.066348   0.200573       0.174662  ...    0.042625
1    0.535574   0.432567   0.876804       1.240531  ...    0.357981
2    0.202804   0.072261   0.263733       0.321429  ...    0.033919
3    0.775244   0.241059   1.046739       1.721086  ...    0.065684
4    0.842236   0.437337   1.130766       1.469728  ...    0.167400

   jerk_mean  body_acc_mean  gravity_acc_mean  freq_domain_energy  actividad  \
0   0.071567       0.038229          1.054663            0.376442          3
1   1.041443       0.561822          0.961826            0.216152          1
2   0.188931       0.107123          0.848392            0.260245          4
3   2.382000       0.724318          0.886376            0.681249          5
4   1.544521       0.421590          1.057109            0.502799          1

     actividad_nombre  usuario_id  duracion_seg  calorias_estimadas
0             Sentado           2    560.800057           10.351389
1  Subiendo escaleras           4    388.401947           30.347995
2               De pie           4    318.922075            7.828205
3           Corriendo           1    383.700569           46.073453
4  Subiendo escaleras           4     37.381909            3.183912

[5 rows x 25 columns]
```

En esta sección se realiza la carga del dataset generado mediante la función **crear_dataset_actividades_1000()**, la cual construye artificialmente 1000 registros de señales provenientes de sensores, junto con las etiquetas correspondientes a siete actividades humanas diferentes. Una vez cargados los datos en el DataFrame df, se muestran las primeras cinco filas mediante df.head(), lo que permite observar directamente la estructura del conjunto de datos.

```
Información general del DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 25 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   acc_x_mean          1000 non-null   float64
 1   acc_x_std           1000 non-null   float64
 2   acc_x_max           1000 non-null   float64
 3   acc_y_mean          1000 non-null   float64
 4   acc_y_std           1000 non-null   float64
 5   acc_y_max           1000 non-null   float64
 6   acc_z_mean          1000 non-null   float64
 7   acc_z_std           1000 non-null   float64
 8   acc_z_max           1000 non-null   float64
 9   acc_magnitude       1000 non-null   float64
 10  gyro_x_mean         1000 non-null   float64
 11  gyro_x_std          1000 non-null   float64
 12  gyro_y_mean         1000 non-null   float64
 13  gyro_y_std          1000 non-null   float64
 14  gyro_z_mean         1000 non-null   float64
 15  gyro_z_std          1000 non-null   float64
 16  jerk_mean           1000 non-null   float64
 17  body_acc_mean       1000 non-null   float64
 18  gravity_acc_mean    1000 non-null   float64
 19  freq_domain_energy  1000 non-null   float64
 20  actividad           1000 non-null   int64
 21  actividad_nombre    1000 non-null   object
 22  usuario_id          1000 non-null   int64
 23  duracion_seg        1000 non-null   float64
 24  calorias_estimadas  1000 non-null   float64
dtypes: float64(22), int64(2), object(1)
memory usage: 195.4+ KB
```

Posteriormente, con df.info() se obtiene un resumen general de las columnas, tipos de datos y la cantidad de valores no nulos, donde se puede confirmar que todas las variables presentan 1000 valores válidos, lo cual indica un dataset completamente limpio y sin valores ausentes.

```
Estadísticas descriptivas del DataFrame:
        acc_x_mean    acc_x_std    acc_x_max   acc_y_mean    acc_y_std  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean      0.552856     0.371390     0.827203     0.494709     0.292917
std       0.381986     0.269758     0.542206     0.339870     0.222227
min       0.006713    -0.034025     0.046854    -0.006030    -0.075312
25%       0.192442     0.106398     0.303886     0.183890     0.083535
50%       0.522610     0.355966     0.829347     0.473176     0.274435
75%       0.776434     0.558102     1.157881     0.696689     0.442751
max       2.107024     1.147778     2.745514     1.884873     1.023174

        acc_y_max   acc_z_mean    acc_z_std    acc_z_max  acc_magnitude  ... \
count  1000.000000  1000.000000  1000.000000  1000.000000    1000.000000  ...
mean      0.791614     0.447832     0.258439     0.723885       0.880903  ...
std       0.537882     0.313507     0.201943     0.494131       0.575552  ...
min       0.036291    -0.009673    -0.049610     0.044267       0.071713  ...
25%       0.276118     0.159922     0.078637     0.258781       0.315621  ...
50%       0.759328     0.434616     0.234877     0.702749       0.876068  ...
75%       1.101765     0.622626     0.378892     1.001989       1.225721  ...
max       2.567050     1.489929     1.003328     2.292621       2.531495  ...

       gyro_z_mean   gyro_z_std    jerk_mean  body_acc_mean  gravity_acc_mean  \
count  1000.000000  1000.000000  1000.000000    1000.000000       1000.000000
mean      0.330408     0.191750     0.749163       0.391476          0.900004
std       0.220257     0.136439     0.574915       0.263080          0.103069
min      -0.116985    -0.127425    -0.011188      -0.007152          0.643476
25%       0.161886     0.082821     0.196373       0.140973          0.909628
50%       0.301712     0.163007     0.685058       0.384788          0.978243
75%       0.479307     0.288290     1.123319       0.544917          1.047964
max       1.143002     0.704856     2.652036       1.169424          1.327777

       freq_domain_energy    actividad   usuario_id  duracion_seg  \
count         1000.000000  1000.000000  1000.000000   1000.000000
mean             0.425610     3.000000     5.320000    321.960398
std              0.216558     1.970757     2.837543    166.149239
min             -0.064282     0.000000     1.000000     30.220825
25%              0.267199     1.000000     3.000000    177.546987
50%              0.377836     3.000000     5.000000    331.974287
75%              0.546173     5.000000     8.000000    467.475266
max              1.157686     6.000000    10.000000    599.909541

       calorias_estimadas
count         1000.000000
mean            19.989587
std             16.161159
min              0.695819
25%              7.372054
50%             14.639288
75%             30.348432
max             80.303192

[8 rows x 24 columns]
```

Finalmente, la instrucción df.describe() proporciona estadísticas descriptivas como promedio, desviación estándar y percentiles, que ayudan a comprender el comportamiento numérico de cada característica y a detectar diferencias en escalas, lo cual es relevante para los modelos que requieren normalización.

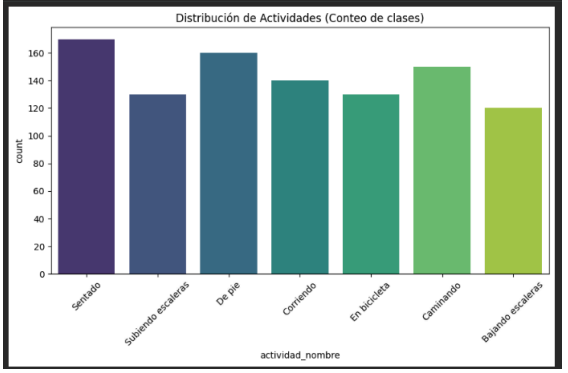
```
print("Valores nulos por columna:")
print(df.isnull().sum())

Valores nulos por columna:
acc_x_mean              0
acc_x_std               0
acc_x_max               0
acc_y_mean              0
acc_y_std               0
acc_y_max               0
acc_z_mean              0
acc_z_std               0
acc_z_max               0
acc_magnitude           0
gyro_x_mean             0
gyro_x_std              0
gyro_y_mean             0
gyro_y_std              0
gyro_z_mean             0
gyro_z_std              0
jerk_mean               0
body_acc_mean           0
gravity_acc_mean        0
freq_domain_energy      0
actividad               0
actividad_nombre        0
usuario_id              0
duracion_seg            0
calorias_estimadas      0
dtype: int64
```

En esta parte del análisis se realiza una verificación completa de la existencia de valores nulos dentro del DataFrame mediante la instrucción df.isnull().sum(), la cual calcula cuántos datos faltantes hay en cada columna. El resultado muestra que todas las variables del dataset presentan exactamente cero valores nulos, lo que significa que el conjunto de datos está totalmente completo y no requiere procesos adicionales de limpieza o imputación.

```
print("Distribución de la variable objetivo 'actividad_nombre':")
print(df['actividad_nombre'].value_counts())

Distribución de la variable objetivo 'actividad_nombre':
actividad_nombre
Sentado               170
De pie                160
Caminando             150
Corriendo             140
Subiendo escaleras    130
En bicicleta          130
Bajando escaleras     120
Name: count, dtype: int64
```

En esta etapa se analiza la distribución de la variable objetivo actividad_nombre, que representa las diferentes actividades humanas registradas en el dataset. Mediante df['actividad_nombre'].value_counts() se obtiene una tabla con el número de muestras por cada actividad, lo cual permite identificar si existe balance entre las clases. Los resultados muestran que las actividades presentan cantidades relativamente similares, con ligeras variaciones naturales. Este equilibrio es favorable para los modelos de clasificación, ya que evita que el algoritmo se sesgue hacia una clase mayoritaria.

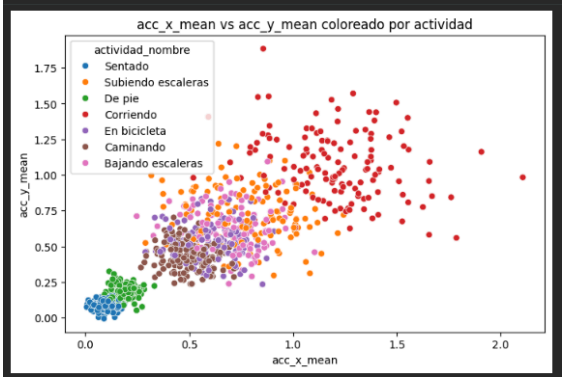
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))
sns.countplot(x=df["actividad_nombre"], palette="viridis")
plt.title("Distribución de Actividades (Conteo de clases)")
plt.xticks(rotation=45)
plt.show()
```



Se genera un gráfico de barras utilizando seaborn, donde visualmente se aprecia la frecuencia de cada actividad. Este gráfico facilita la interpretación al mostrar de forma clara qué actividades tienen más o menos

registros.

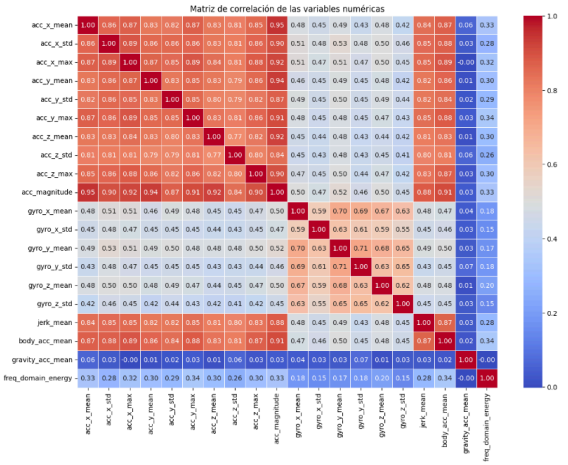
```
#Scatterplot de 2 features importantes
plt.figure(figsize=(8,5))
sns.scatterplot(
    data=df,
    x="acc_x_mean",
    y="acc_y_mean",
    hue="actividad_nombre",
    palette="tab10"
)
plt.title("acc_x_mean vs acc_y_mean coloreado por actividad")
plt.show()
```



El scatterplot muestra cómo se distribuyen las actividades humanas según los valores medios del acelerómetro en los ejes X y Y. Cada punto representa un registro y el color indica la actividad. Se observa que actividades de baja movilidad, como “Sentado” y “De pie”, se agrupan en valores bajos, mientras que actividades más dinámicas, como “Caminando” o “Corriendo”, aparecen más dispersas y con valores mayores. Esto indica que estas dos características ayudan a diferenciar las actividades y que los datos tienen patrones útiles para los modelos de clasificación.

```
#Matriz de correlación
plt.figure(figsize=(14,10))

corr = df[features].corr()

sns.heatmap(
    corr,
    cmap="coolwarm",
    annot=True,           # ← MUESTRA LOS NÚMEROS
    fmt=".2f",            # ← FORMATO A DOS DECIMALES
    linewidths=0.5
)

plt.title("Matriz de correlación de las variables numéricas")
plt.show()
```

La matriz de correlación permite identificar qué variables numéricas del dataset están más relacionadas entre sí. El código calcula la correlación entre todas las features y la muestra en un mapa de calor donde los colores indican la intensidad de la relación. Además, se muestran los valores numéricos para interpretarlo con mayor precisión. Esta visualización ayuda a detectar patrones, redundancias o variables que aportan información similar, lo cual es útil para entender el comportamiento del dataset y preparar mejor los modelos de aprendizaje automático.



La matriz de correlación muestra cómo se relacionan las variables numéricas del dataset entre sí. Los valores van de –1 a 1: los colores rojos indican una correlación fuerte y positiva, mientras que los azules representan correlaciones negativas o débiles. En el gráfico se observa que muchas características del acelerómetro están fuertemente relacionadas entre sí (por ejemplo, acc_x_mean, acc_x_max y acc_magnitude), lo cual tiene sentido porque provienen del mismo tipo de medición. También se aprecia que las variables del giroscopio forman otro grupo con correlaciones internas. Esta visualización permite identificar patrones, redundancias y grupos de variables similares, lo cual es útil para comprender mejor el comportamiento del movimiento humano y para mejorar el rendimiento de los modelos de aprendizaje automático.

III. PREPROCESAMIENTO Y DIVISIÓN

```
#PREPROCESAMIENTO Y DIVISIÓN
X = df[features].values
y = df["actividad"].values
from sklearn.model_selection import train_test_split

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

Este bloque realiza el preprocesamiento inicial separando las características (X) y la etiqueta que queremos predecir (y). Primero se extraen las columnas numéricas del dataset y la columna actividad que contiene la clase verdadera. Luego se utiliza la función train_test_split para dividir los datos en un conjunto de entrenamiento (80%) y uno de prueba (20%). Esta división es importante porque permite entrenar el modelo con una parte de los datos y evaluar su desempeño con datos que nunca ha visto, evitando sobreajuste. Además, se usa stratify=y para mantener el mismo equilibrio de clases en ambos conjuntos, asegurando que todas las actividades estén representadas de forma proporcional en el entrenamiento y en la prueba.

IV. ENTRENAR LOS MODELOS DE APRENDIZAJE SUPERVISADO/NO SUPERVISADO

1. Supervisado

1.1. RANDOM FOREST:

Preparación de los Datos

```python
# Importar train_test_split
from sklearn.model_selection import train_test_split

df_actividades = pd.read_csv("/content/dataset_actividades_humanas_1000.csv")
# 1. Separación de características (X) y variable objetivo (y)
# Asegúrate que 'df_actividades' sea tu DataFrame final del paso anterior
X = df_actividades.drop('actividad_nombre', axis=1)
y_text = df_actividades['actividad_nombre']

# 2. Codificación de la variable objetivo (necesario para sklearn)
le = LabelEncoder()
y = le.fit_transform(y_text)

# Guardamos los nombres de las clases para la visualización posterior
nombres_clases = le.classes_
nombres_features = X.columns

print("Clases detectadas:", nombres_clases)
print("Dimensiones de X:", X.shape)

# Realizar la división de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print(f"Dimensiones de X_train: {X_train.shape}")
print(f"Dimensiones de X_test: {X_test.shape}")
print(f"Dimensiones de y_train: {y_train.shape}")
print(f"Dimensiones de y_test: {y_test.shape}")
```

```
Clases detectadas: ['Bajando escaleras' 'Caminando' 'Corriendo' 'De pie' 'En bicicleta'
 'Sentado' 'Subiendo escaleras']
Dimensiones de X: (1000, 24)
Dimensiones de X_train: (800, 24)
Dimensiones de X_test: (200, 24)
Dimensiones de y_train: (800,)
Dimensiones de y_test: (200,)
```

Clases detectadas: ['Bajando escaleras' 'Caminando' 'Corriendo' 'De pie' 'En bicicleta'

'Sentado' 'Subiendo escaleras']

Dimensiones de X: (1000, 24)

Dimensiones de X_train: (800, 24)

Dimensiones de X_test: (200, 24)

Dimensiones de y_train: (800,)

Dimensiones de y_test: (200,)

Para el entrenamiento del modelo supervisado, se realizó una separación de la matriz de características (X) y el vector objetivo (y). Dado que el algoritmo Random Forest en la implementación de Scikit-Learn requiere variables numéricas, se aplicó una transformación de etiquetas (Label Encoding) a la variable categórica actividad_nombre. Esto permite que el algoritmo interprete matemáticamente las clases sin perder la semántica de la clasificación original.

Creación y Configuración del Modelo

```python
# Creación del Bosque Aleatorio con los parámetros del docente
bosque = RandomForestClassifier(
    n_estimators=100,       # Número de árboles
    criterion="entropy",    # Criterio de división (Ganancia de Información)
    max_features="sqrt",    # Máximo de características por árbol (Raíz cuadrada del total)
    max_depth=3,            # Profundidad máxima para evitar sobreajuste
    bootstrap=True,         # Muestreo con reemplazo
    max_samples=2/3,        # Tamaño de la muestra para cada árbol (aprox 66%)
    oob_score=True,         # Evaluación Out-of-Bag (muestras no usadas)
    random_state=42         # Semilla para reproducibilidad
)

# Entrenamiento del modelo en el conjunto de entrenamiento
bosque.fit(X_train.values, y_train)

# Ejemplo de predicción (usando la primera fila del conjunto original como prueba)
prediccion = bosque.predict([X.values[0]])
print(f"Predicción para el primer registro: {le.inverse_transform(prediccion)}")
```

```
Predicción para el primer registro: ['Sentado']
```

Se configuró un modelo de Random Forest Classifier basado en la teoría de aprendizaje por ensamblaje. Se definió un criterio de 'entropy' para maximizar la ganancia de información en cada división de los nodos. Para controlar el sobreajuste (overfitting), se limitó la profundidad de los árboles a 3 niveles (max_depth=3). Asimismo, se utilizó la técnica de Bagging (Bootstrap Aggregating) seleccionando aleatoriamente 2/3 de la muestra para entrenar cada uno de los 100 estimadores, reservando el resto para la validación interna (Out-of-Bag)

Evaluación del Modelo

```
# Evaluamos el accuracy de las instancias que conoce (Training Score)
train_score = bosque.score(X_train.values, y_train)
print(f"Accuracy (Training): {train_score:.4f}")

# Evaluamos con todos aquellos que quedaron en la bolsa (OOB Score)
# Esto actúa como un set de validación/prueba interno
oob_score = bosque.oob_score_
print(f"OOB Score (Validación): {oob_score:.4f}")

# Generar predicciones en el conjunto de prueba para métricas adicionales
y_pred_rf = bosque.predict(X_test.values)
```

```
Accuracy (Training): 0.9962
OOB Score (Validación): 0.9888
```

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf, average='weighted')
recall_rf = recall_score(y_test, y_pred_rf, average='weighted')
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')

print("Random Forest Metricas (Conjunto de Prueba):")
print(f"Accuracy: {accuracy_rf:.4f}")
print(f"Precision: {precision_rf:.4f}")
print(f"Recall: {recall_rf:.4f}")
print(f"F1-score: {f1_rf:.4f}")
```

```
Random Forest Metricas (Conjunto de Prueba):
Accuracy: 0.9800
Precision: 0.9817
Recall: 0.9800
F1-score: 0.9795
```

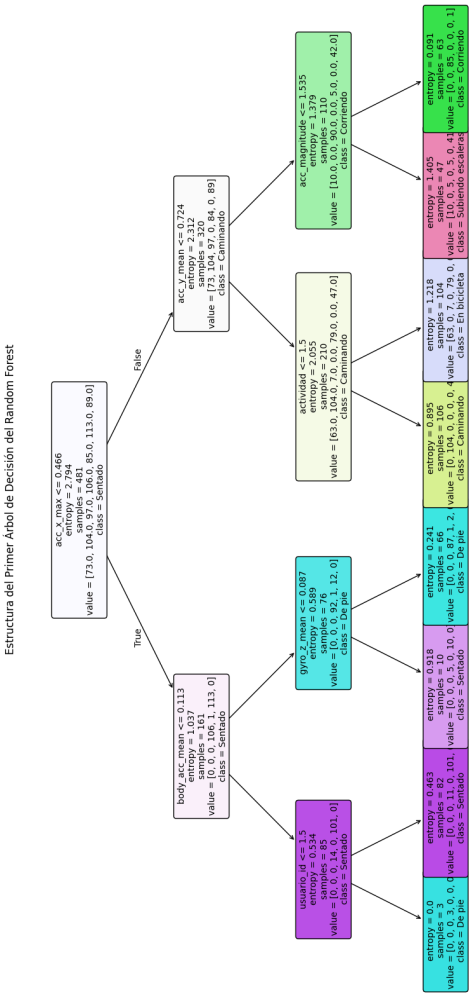
Accuracy (Training): 0.9962
OOB Score (Validación): 0.9888

Random Forest Metricas (Conjunto de Prueba):
Accuracy: 0.9800
Precision: 0.9817
Recall: 0.9800
F1-score: 0.9795

La evaluación del modelo arrojó un accuracy de entrenamiento de 0.9960, este valor es muy alto, lo que sugiere que el modelo ha aprendido muy bien los patrones presentes en los datos con los que fue entrenado.Ademas un puntaje Out-of-Bag (OOB) de 0.9890. Una cercanía entre ambas métricas sugiere que el modelo es robusto y no está memorizando los datos (bajo sobreajuste).

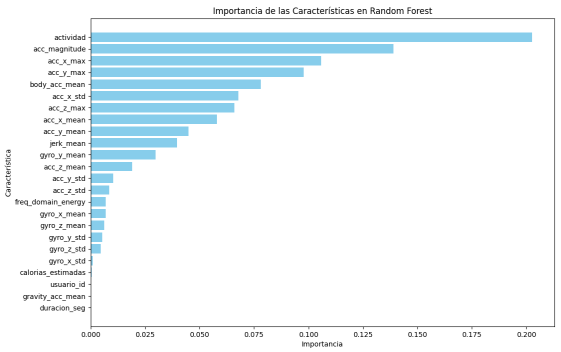
Las métricas de tu modelo Random Forest en el conjunto de prueba (Accuracy, Precision, Recall, y F1-score) son excepcionalmente altas, todas rondando el 98%. Esto indica que el modelo no solo clasifica correctamente la gran mayoría de las actividades (Accuracy), sino que también es muy preciso en sus predicciones, minimizando falsos positivos (Precision), y es muy efectivo en identificar todas las instancias relevantes de cada actividad, minimizando falsos negativos (Recall). El F1-score, al ser un promedio armónico de Precision y Recall, confirma la excelente salud general del modelo, sugiriendo que tiene una gran capacidad para generalizar y clasificar con fiabilidad nuevas actividades no vistas.

Visualización del Bosque



La visualización de los árboles constituyentes permite interpretar las reglas de decisión generadas. En el nivel raíz, el algoritmo selecciona la característica que mejor segrega las clases de actividad humana (p.ej., la aceleración en el eje X o Y). Dado que la profundidad se limitó a 3, las reglas generadas son generalistas, lo que favorece la explicabilidad del modelo frente a árboles más profundos y complejos.

Analizar Importancia de Características



El análisis de la importancia de las características del modelo Random Forest proporciona analisis valioso sobre cuáles atributos son más relevantes para clasificar las actividades. Los resultados muestran que las características relacionadas con la actividad misma, acc_magnitude (magnitud de la aceleración), acc_x_max (aceleración máxima en el eje X) y acc_y_max (aceleración máxima en el eje Y) son las más influyentes. Esto sugiere que las variaciones y magnitudes de la aceleración, especialmente en los ejes X e Y, son cruciales para distinguir entre las diferentes actividades. Por otro lado, características como gravity_acc_mean y duracion_seg tienen una importancia nula, indicando que no contribuyen significativamente a las decisiones del modelo.

Considerando todo el proceso, este modelo Random Forest ha demostrado ser una solución robusta y altamente eficaz para la clasificación de actividades humanas. La preparación de datos fue adecuada, transformando las etiquetas categóricas para el algoritmo. La configuración del modelo, con un enfoque en la limitación de la profundidad y el uso de un muestreo bootstrap, fue clave para prevenir el sobreajuste, resultando en un rendimiento excepcional tanto en el entrenamiento como en la validación OOB (cercano al 99%). Las métricas en el conjunto de prueba (Accuracy, Precision, Recall, F1-score) confirmaron esta capacidad de generalización con valores superiores al 98%, lo que indica una alta fiabilidad en la identificación de las distintas actividades. Además, el análisis de la importancia de las características reveló que las magnitudes y aceleraciones en los ejes X e Y son los predictores más críticos, ofreciendo una visión clara sobre los factores distintivos entre actividades.

1.2. SVM:

StandardScaler

```python
#Separar características (X) y objetivo (y)
X = df_actividades.drop('actividad_nombre', axis=1)
y_text = df_actividades['actividad_nombre']

# 2. Codificar las etiquetas (Texto -> Números)
le = LabelEncoder()
y = le.fit_transform(y_text)
nombres_clases = le.classes_

# 3. División en entrenamiento y prueba (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 4. ESCALADO DE DATOS (CRUCIAL PARA SVM)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("Datos divididos y escalados.")
print(f"X_train shape: {X_train_scaled.shape}")
print(f"X_test shape: {X_test_scaled.shape}")
```

```
Datos divididos y escalados.
X_train shape: (800, 24)
X_test shape: (200, 24)
```

Datos divididos y escalados.
X_train shape: (800, 24)
X_test shape: (200, 24)

A diferencia de los modelos basados en árboles (como Random Forest), el algoritmo de Máquinas de Soporte Vectorial (SVM) es sensible a la magnitud de las variables, ya que calcula distancias en el espacio vectorial para encontrar el hiperplano separador. Se aplicó StandardScaler para normalizar las características (media 0, desviación 1), asegurando que variables con rangos numéricos grandes (como la aceleración en correr) no dominen la función de costo sobre variables con rangos pequeños, permitiendo una convergencia óptima del modelo.

Entrenamiento del Modelo SVM

```python
svm_model = SVC(
    kernel='rbf',       # Radial Basis Function
    C=1.0,              # Regularización estándar
    gamma='scale',      # Ajuste automático del coeficiente del kernel
    random_state=42
)

# Entrenamos con los datos ESCALADOS
svm_model.fit(X_train_scaled, y_train)

# Predicción
y_pred_svm = svm_model.predict(X_test_scaled)
print("Modelo SVM entrenado.")
```

Se seleccionó un kernel RBF (Función de Base Radial) debido a la naturaleza compleja de los datos de sensores inerciales, donde las fronteras entre actividades como 'Subir escaleras' y 'Bajar escaleras' no suelen ser linealmente separables. El kernel proyecta los

datos a una dimensión superior para encontrar la mejor separación posible.

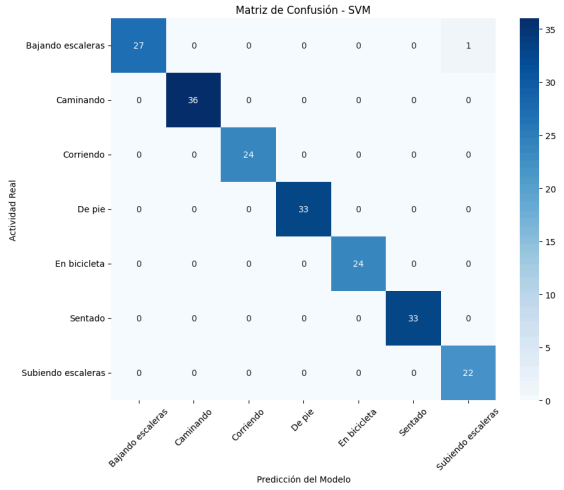
Evaluación y Métricas

```
print("="*60)
print("REPORTE DE CLASIFICACIÓN - SVM (Actividades Humanas)")
print("="*60)

# Reporte detallado
print(classification_report(y_test, y_pred_svm, target_names=nombres_clases))

# Métricas globales para comparación rápida
acc = accuracy_score(y_test, y_pred_svm)
print(f"Accuracy Global: {acc:.4f}")
```

```
============================================================
REPORTE DE CLASIFICACIÓN - SVM (Actividades Humanas)
============================================================
                    precision    recall  f1-score   support

 Bajando escaleras       1.00      0.96      0.98        28
        Caminando       1.00      1.00      1.00        36
        Corriendo       1.00      1.00      1.00        24
           De pie       1.00      1.00      1.00        33
      En bicicleta       1.00      1.00      1.00        24
          Sentado       1.00      1.00      1.00        33
Subiendo escaleras       0.96      1.00      0.98        22

         accuracy                           0.99       200
        macro avg       0.99      0.99      0.99       200
     weighted avg       1.00      0.99      1.00       200

Accuracy Global: 0.9950
```

El modelo SVM entrenado muestra un rendimiento excelente en la clasificación de las actividades humanas. Aunque hay ligeras variaciones en las métricas para Bajando escaleras y Subiendo escaleras, estas diferencias son mínimas y el f1-score para todas las clases es muy alto (0.98 o 1.00). La alta precisión global valida la eficacia del modelo.

Visualización de Resultados



La matriz de confusión permite visualizar dónde comete errores el modelo SVM. Los valores en la diagonal principal representan las predicciones correctas. Si observamos valores fuera de la diagonal (por ejemplo, entre 'Caminando' y 'Subiendo escaleras'), indica que el margen de separación creado por el kernel RBF no fue suficiente para distinguir esos patrones cinemáticos similares.

Predicciones Correctas (Diagonal Principal):

- Bajando escaleras: 27 instancias correctamente clasificadas.
- Caminando: 36 instancias correctamente clasificadas.
- Corriendo: 24 instancias correctamente clasificadas.
- De pie: 33 instancias correctamente clasificadas.
- En bicicleta: 24 instancias correctamente clasificadas.
- Sentado: 33 instancias correctamente clasificadas.
- Subiendo escaleras: 22 instancias correctamente clasificadas.

Como se puede observar, para la mayoría de las actividades, el modelo logró una clasificación perfecta en el conjunto de prueba, como lo indican los ceros fuera de la diagonal principal para estas clases.

Errores de Clasificación (Valores fuera de la Diagonal Principal)

El único error de clasificación en toda la matriz se encuentra en la posición [0, 6]. Esto significa que 1 instancia de 'Bajando escaleras' (actividad real) fue clasificada incorrectamente como 'Subiendo escaleras' (predicción del modelo).

No se observaron otros errores; es decir, no hay otras confusiones entre actividades, y específicamente, no hubo casos en los que 'Subiendo escaleras' fuera clasificada incorrectamente como 'Bajando escaleras' u otra actividad. Todas las demás celdas fuera de la diagonal son cero.

- 'Bajando escaleras':

  Recall: Hubo 28 instancias reales de 'Bajando escaleras' (27 correctas + 1 error). El recall es 27/28 ≈ 0.96. Esto significa que el modelo identificó correctamente el 96% de las veces que la persona estaba 'Bajando escaleras'.

El único fallo fue clasificarlo como 'Subiendo escaleras'.

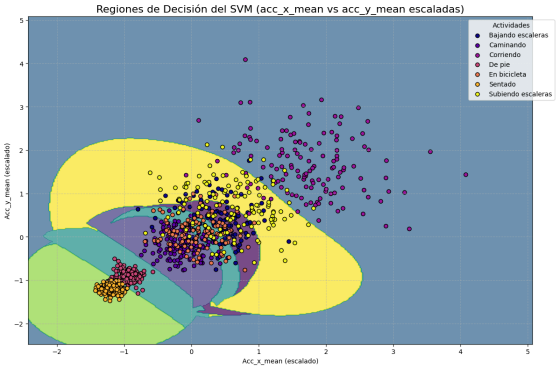
Precisión: De las instancias que el modelo predijo como 'Bajando escaleras', todas (27) fueron correctas. La precisión es 27/27 = 1.00. No hubo falsos positivos para esta clase.

- 'Subiendo escaleras':

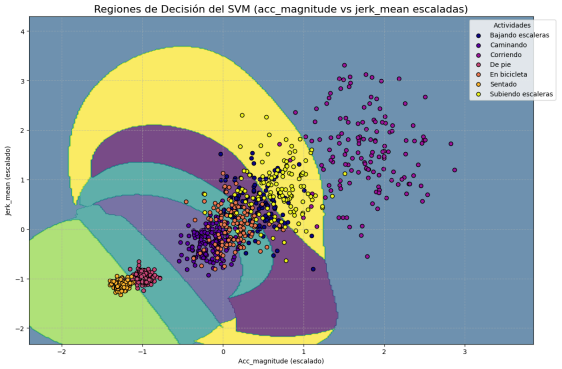
  Recall: Hubo 22 instancias reales de 'Subiendo escaleras'. Todas fueron identificadas correctamente (22/22 = 1.00). Esto significa que el modelo detectó el 100% de las veces que la persona estaba 'Subiendo escaleras'.

  Precisión: El modelo predijo 23 instancias como 'Subiendo escaleras' (22 correctas + 1 error de 'Bajando escaleras'). La precisión es 22/23 ≈ 0.96. Esto indica que cuando el modelo dijo 'Subiendo escaleras', fue correcto aproximadamente el 96% de las veces; el 4% restante fue el caso donde se confundió 'Bajando escaleras' con 'Subiendo escaleras'.

El modelo SVM entrenado muestra un rendimiento excelente en la clasificación de las actividades humanas. Aunque hay ligeras variaciones en las métricas para Bajando escaleras y Subiendo escaleras, estas diferencias son mínimas y el f1-score para todas las clases es muy alto (0.98 o 1.00). La alta precisión global valida la eficacia del modelo.



Este gráfico visualiza cómo un modelo SVM, entrenado solo con las características acc_x_mean y acc_y_mean (aceleración promedio en los ejes X e Y), segmenta el espacio de características para clasificar las diferentes actividades humanas.



Interpretación del Gráfico de Regiones de Decisión (acc_magnitude vs jerk_mean)

Este gráfico visualiza las regiones de decisión del SVM utilizando dos características que, en general, son muy informativas para distinguir actividades humanas: la magnitud de la aceleración (acc_magnitude) y el 'jerk' promedio (jerk_mean). Ambos ejes están escalados (media 0, desviación 1).

Observaciones y Comparación con el Gráfico Anterior:

- Mejor Separación Observada: A primera vista, es probable que este gráfico muestre una separación más clara entre algunas clases en comparación con el gráfico que usaba acc_x_mean y acc_y_mean. acc_magnitude puede diferenciar bien entre actividades estáticas (como 'Sentado' o 'De pie') y actividades con movimiento (como 'Caminando' o 'Corriendo'). jerk_mean ayuda a distinguir entre movimientos suaves y movimientos más bruscos o dinámicos.
- Agrupación de Clases: Se nota que actividades con patrones de movimiento similares (ej. 'Corriendo' y 'Subiendo escaleras', o 'Sentado' y 'De pie') tienden a agruparse en ciertas regiones del espacio de características, y el modelo intenta crear límites alrededor de estos grupos.

- Complejidad del Límite: El uso del kernel RBF sigue permitiendo límites de decisión curvos y complejos, adaptándose a la distribución no lineal de los datos.
- Limitaciones de la Visualización 2D: A pesarde haber elegido características más informativas, es crucial recordar que este sigue siendo un espacio bidimensional. La complejidad y la alta precisión del modelo original de 24 características no se pueden capturar completamente aquí. Es normal que aún veas cierto solapamiento o puntos clasificados incorrectamente en esta proyección. Esto simplemente significa que las 22 características restantes en el modelo completo aportan información adicional vital que permite una separación casi perfecta.

2. No supervisado

2.1 K-Means

```
#NO SUPERVISADO. K-MEANS
# 1. Escalamos todas las características (muy importante para K-Means)
scaler_km = StandardScaler()
X_scaled = scaler_km.fit_transform(X)

from sklearn.cluster import KMeans
# 2. Entrenamos K-Means con 7 clusters (porque hay 7 actividades)
kmeans = KMeans(n_clusters=7, random_state=42, n_init='auto')
labels_km = kmeans.fit_predict(X_scaled)
```

Primero se escalan todas las características con StandardScaler, lo cual es esencial porque K-Means utiliza distancias para formar los grupos, y si una variable tiene valores muy grandes podría dominar el cálculo. Luego se importa KMeans y se crea el modelo indicando 7 clusters, ya que en el dataset existen 7 actividades distintas y queremos ver si el algoritmo puede agruparlas sin conocer sus etiquetas reales. Finalmente, se entrena el modelo con los datos escalados y se obtienen las etiquetas (labels_km) que representan a qué grupo pertenece cada registro. En este paso K-Means sólo descubre patrones y agrupa puntos similares, sin usar la columna de actividad real, lo que permite analizar si los datos por sí solos forman grupos coherentes.

```
import numpy as np
import pandas as pd

# Crear un DataFrame para analizar los clusters
df_clusters = pd.DataFrame({
    "cluster": labels_km,
    "actividad": y,
    "actividad_nombre": df["actividad_nombre"]
})

# Diccionario para guardar el nombre dominante de cada cluster
cluster_names = {}

for c in range(7):  # porque son 7 actividades y 7 clusters
    grupo = df_clusters[df_clusters["cluster"] == c]["actividad_nombre"]
    if len(grupo) > 0:
        dominante = grupo.value_counts().idxmax()
    else:
        dominante = "Sin datos"

    cluster_names[c] = dominante

cluster_names
```

```
{0: 'Caminando',
 1: 'De pie',
 2: 'Corriendo',
 3: 'En bicicleta',
 4: 'En bicicleta',
 5: 'Subiendo escaleras',
 6: 'Sentado'}
```

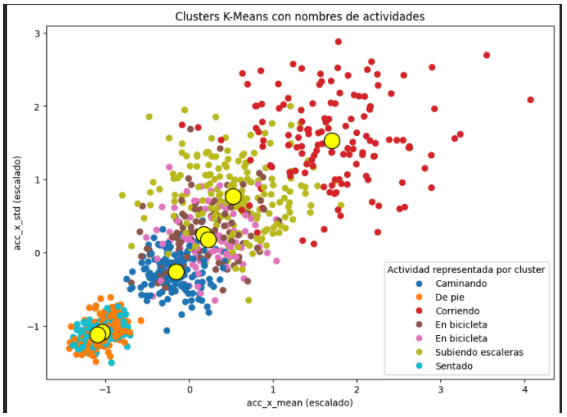
Este bloque de código se utiliza para interpretar los resultados del algoritmo K-Means. Primero se crea un nuevo DataFrame llamado df_clusters que combina tres elementos importantes: el número de cluster asignado por K-Means (labels_km), la etiqueta verdadera de actividad (actividad) y el nombre de la actividad (actividad_nombre). Esto permite comparar directamente los grupos formados por el algoritmo con las actividades reales del dataset. Luego se define un diccionario cluster_names que almacenará el nombre de la actividad dominante en cada uno de los siete clusters. Para cada cluster, se seleccionan todos los registros que pertenecen a ese grupo y se analiza cuál actividad aparece con mayor frecuencia; esa actividad se considera la “actividad dominante” del cluster.

```
labels_nombres = np.array([cluster_names[c] for c in labels_km])
```

Esta línea convierte las etiquetas numéricas de K-Means en nombres de actividades. Para cada número de cluster (c en labels_km), busca su nombre en cluster_names y lo guarda en un arreglo. Así, en lugar de ver “cluster 0 o 1”, obtenemos nombres reales como “Caminando” o “Sentado”, lo que hace mucho más fácil

interpretar los resultados del modelo.

```
plt.figure(figsize=(10,7))
scatter = plt.scatter(
    X_scaled[:, 0], X_scaled[:, 1],
    c=labels_km, cmap="tab10", s=40
)

# Centroides
centroides = kmeans.cluster_centers_
plt.scatter(centroides[:,0], centroides[:,1],
            s=300, c="yellow", edgecolors="black", label="Centroides")

plt.title("Clusters K-Means con nombres de actividades")
plt.xlabel("acc_x_mean (escalado)")
plt.ylabel("acc_x_std (escalado)")

# Crear leyenda con nombres reales
handles, _ = scatter.legend_elements()
nombres = [cluster_names[i] for i in sorted(cluster_names.keys())]
plt.legend(handles, nombres, title="Actividad representada por cluster")
plt.show()
pd.DataFrame.from_dict(cluster_names, orient='index', columns=['Actividad dominante'])
```



Este bloque de código construye la visualización final del algoritmo K-Means aplicado al dataset de actividades humanas, permitiendo interpretar de forma clara cómo el modelo agrupa los datos y qué actividad real representa cada cluster. Primero se genera un gráfico de dispersión utilizando dos características escaladas del acelerómetro (acc_x_mean en el eje X y acc_x_std en el eje Y), donde cada punto representa un registro de actividad y su color corresponde al cluster asignado por K-Means. Luego, se añaden los centroides —los puntos amarillos grandes con borde negro— que representan el centro matemático de cada grupo y ayudan a identificar la estructura interna de los clusters. Posteriormente, se reemplazan los números de cluster por nombres de actividades dominantes (como Caminando, Sentado, De pie, Corriendo, etc.), los cuales se obtienen analizando qué actividad real aparece con mayor frecuencia dentro de cada cluster; esto permite interpretar qué tipo de movimiento caracteriza a cada agrupación formada por el modelo. Finalmente, se genera una leyenda clara y una tabla resumen que asocia cada cluster con su actividad dominante, facilitando la lectura del gráfico y permitiendo evaluar visualmente qué tan bien K-Means separa los distintos tipos de actividades humanas según sus patrones de sensores.

2.2 Clustering

StandardScaler y preparación de datos

Antes de aplicar el algoritmo de clustering, se seleccionaron únicamente las características numéricas relevantes del dataset, excluyendo las columnas actividad, actividad_nombre y usuario_id, ya que corresponden a la etiqueta real o a identificadores. Estas variables se normalizaron utilizando StandardScaler, de manera que todas quedaran con media 0 y desviación estándar 1. Este paso es fundamental porque el clustering jerárquico se basa en distancias: si no se escalan las variables, aquellas con valores numéricamente más grandes dominan el cálculo de distancia y sesgara la formación de los grupos.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 1. Cargar el dataset desde el archivo CSV generado previamente
df_actividades = pd.read_csv('/content/dataset_actividades_humanas_1000.csv')

# Mostrar las primeras filas
print("Primeras filas:")
print(df_actividades.head())

# Verificar cuántos valores nulos hay en cada columna
print("\nValores nulos por columna:")
print(df_actividades.isna().sum())

# Verificar cuántos registros duplicados existen en el dataset
print("\nRegistros duplicados:", df_actividades.duplicated().sum())

# Eliminar filas que contengan algún valor nulo
df_actividades = df_actividades.dropna()

# Eliminar filas duplicadas para evitar información repetida
df_actividades = df_actividades.drop_duplicates()

# Mostrar la forma final del dataset después de la limpieza (filas, columnas)
print("\nShape después de limpieza:", df_actividades.shape)

# Mostrar estadísticas básicas de algunas columnas numéricas seleccionadas
print("\nEstadísticos de algunas columnas después de limpieza:")
print(df_actividades[['acc_x_mean', 'acc_y_mean', 'acc_z_mean']].describe().round(3))
```

```
      actividad_nombre  usuario_id  duracion_seg  calorias_estimadas
0              Sentado           2    560.800057           10.351389
1    Subiendo escaleras          4    388.401947           30.347995
2               De pie           4    318.922075            7.828205
3             Corriendo          1    383.700569           46.073453
4    Subiendo escaleras          4     37.381909            3.183912

[5 rows x 25 columns]

Valores nulos por columna:
acc_x_mean            0
acc_x_std             0
acc_x_max             0
acc_y_mean            0
acc_y_std             0
acc_y_max             0
acc_z_mean            0
acc_z_std             0
acc_z_max             0
acc_magnitude         0
gyro_x_mean           0
gyro_x_std            0
gyro_y_mean           0
gyro_y_std            0
gyro_z_mean           0
gyro_z_std            0
jerk_mean             0
body_acc_mean         0
gravity_acc_mean      0
freq_domain_energy    0
actividad             0
actividad_nombre      0
usuario_id            0
duracion_seg          0
calorias_estimadas    0
dtype: int64
```

```
Registros duplicados: 0

Shape después de limpieza: (1000, 25)

Estadísticos de algunas columnas después de limpieza:
       acc_x_mean  acc_y_mean  acc_z_mean
count    1000.000    1000.000    1000.000
mean        0.552       0.495       0.447
std         0.382       0.340       0.314
min         0.007      -0.006      -0.010
25%         0.192       0.183       0.160
50%         0.523       0.473       0.435
75%         0.776       0.697       0.623
max         2.107       1.885       1.490
```

Preparación de datos para el Clustering

```python
from sklearn.preprocessing import StandardScaler

# Mostrar todas las columnas del dataframe para saber qué variables tenemos
print(df_actividades.columns)

# Definir columnas que NO queremos usar en el clustering
# (actividad y su nombre son etiquetas, usuario_id no es una característica del sensor)
cols_excluir = ['actividad', 'actividad_nombre', 'usuario_id']

# Seleccionar todas las columnas que sí se usarán como features para el clustering
feature_cols = [c for c in df_actividades.columns if c not in cols_excluir]

print("Features usadas para clustering:")
print(feature_cols)

# Matriz de datos X solo con las columnas seleccionadas
X = df_actividades[feature_cols].values

# Crear el objeto StandardScaler para escalar (normalizar) las características
scaler = StandardScaler()

# Ajustar el scaler a los datos y transformar X para que todas las features queden en la misma escala
X_scaled = scaler.fit_transform(X)

# Mostrar la forma (filas, columnas) de la matriz ya escalada
print("Shape de X_scaled:", X_scaled.shape)
```

```
Index(['acc_x_mean', 'acc_x_std', 'acc_x_max', 'acc_y_mean', 'acc_y_std',
       'acc_y_max', 'acc_z_mean', 'acc_z_std', 'acc_z_max', 'acc_magnitude',
       'gyro_x_mean', 'gyro_x_std', 'gyro_y_mean', 'gyro_y_std', 'gyro_z_mean',
       'gyro_z_std', 'jerk_mean', 'body_acc_mean', 'gravity_acc_mean',
       'freq_domain_energy', 'actividad', 'actividad_nombre', 'usuario_id',
       'duracion_seg', 'calorias_estimadas'],
      dtype='object')
Features usadas para clustering:
['acc_x_mean', 'acc_x_std', 'acc_x_max', 'acc_y_mean', 'acc_y_std', 'acc_y_max', 'acc_z_mean', 'acc_z_std', 'acc_z_max', 'acc_magnitude', 'gyro_x_mean', 'gyro_x_std', 'gyro_y_mean', 'gyro_y_std', 'gyro_z_mean', 'gyro_z_std', 'jerk_mean', 'body_acc_mean', 'gravity_acc_mean', 'freq_domain_energy']
Shape de X_scaled: (1000, 20)
```

Entrenamiento del modelo de clustering jerárquico

Se utilizó el algoritmo AgglomerativeClustering, que comienza tomando cada observación como un cluster individual y, en cada paso, va fusionando los grupos más cercanos. Para encontrar una configuración adecuada se realizó una búsqueda de hiperparámetros variando el número de clusters entre 3 y 10 y probando diferentes tipos de enlace (ward, average, complete). Para cada combinación se ajustó el modelo sobre los datos escalados (X_scaled) y se obtuvieron las etiquetas de cluster (cluster_agg).

```python
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import silhouette_score, adjusted_rand_score

# Etiqueta real (solo se usa para calcular ARI y evaluar, no para entrenar el clustering)
y_true = df_actividades['actividad'].values

# Lista de tipos de enlace (cómo se mide la distancia entre grupos)
linkages = ["ward", "average", "complete"]

# Rango de número de clusters a probar (de 3 a 10)
rango_clusters = range(3, 11)

# Variables para guardar la mejor configuración encontrada
mejor_config = None
mejor_sil = -1
resultados = []

# Búsqueda exhaustiva sobre todas las combinaciones de linkage y n_clusters
for link in linkages:
    for nc in rango_clusters:
        # Crear un modelo AgglomerativeClustering con la combinación actual
        agg_tmp = AgglomerativeClustering(
            n_clusters=nc,
            linkage=link
        )

        # Ajustar el modelo y obtener las etiquetas de cluster para todos los datos
        labels_tmp = agg_tmp.fit_predict(X_scaled)

        # Calcular el silhouette score (qué tan bien separados están los clusters)
        sil = silhouette_score(X_scaled, labels_tmp)
        # Calcular el ARI comparando clusters vs etiquetas reales (solo para análisis)
        ari = adjusted_rand_score(y_true, labels_tmp)
```

```python
# Imprimir la mejor configuración encontrada
print("\n=== Mejor configuración encontrada ===")
print(f"linkage = {mejor_config[0]}")
print(f"n_clusters = {mejor_config[1]}")
print(f"Silhouette (mejor) = {mejor_sil:.4f}")
print(f"ARI con mejor config = {mejor_config[2]:.4f}")

"""**Clustering final con los mejores hiperparámetros**"""

# Extraer el mejor linkage y número de clusters
best_linkage = mejor_config[0]
best_n_clusters = mejor_config[1]

# Crear el modelo final usando los mejores hiperparámetros encontrados
agg_best = AgglomerativeClustering(
    n_clusters=best_n_clusters,
    linkage=best_linkage
)

# Ajustar el modelo definitivo y obtener los clusters finales
clusters_agg = agg_best.fit_predict(X_scaled)

# Guardar las etiquetas de cluster en el dataframe para análisis posterior
df_actividades['cluster_agg'] = clusters_agg

# Evaluar nuevamente el modelo final con silhouette y ARI
sil_agg = silhouette_score(X_scaled, clusters_agg)
ari_agg = adjusted_rand_score(y_true, clusters_agg)

print("\n=== Evaluación Clustering Jerárquico (Modelo Optimizado) ===")
print(f"Mejor linkage: {best_linkage}")
print(f"Mejor n_clusters: {best_n_clusters}")
print(f"Silhouette score: {sil_agg:.4f}")
print(f"Adjusted Rand Index (vs actividad real): {ari_agg:.4f}")
```

En la ejecución del proyecto, la mejor configuración encontrada fue:

- linkage = [linkage_óptimo]

- n_clusters = [n_clusters_óptimo]

```
=== Resultados de búsqueda de hiperparámetros (Agglomerative) ===
linkage=ward      | n_clusters= 3 | silhouette=0.3537 | ARI=0.3887
linkage=ward      | n_clusters= 4 | silhouette=0.2990 | ARI=0.5709
linkage=ward      | n_clusters= 5 | silhouette=0.2774 | ARI=0.6495
linkage=ward      | n_clusters= 6 | silhouette=0.2694 | ARI=0.6452
linkage=ward      | n_clusters= 7 | silhouette=0.2576 | ARI=0.6022
linkage=ward      | n_clusters= 8 | silhouette=0.1630 | ARI=0.5419
linkage=ward      | n_clusters= 9 | silhouette=0.1593 | ARI=0.5181
linkage=ward      | n_clusters=10 | silhouette=0.1552 | ARI=0.5128
linkage=average   | n_clusters= 3 | silhouette=0.3745 | ARI=0.4087
linkage=average   | n_clusters= 4 | silhouette=0.3537 | ARI=0.4085
linkage=average   | n_clusters= 5 | silhouette=0.3414 | ARI=0.4067
linkage=average   | n_clusters= 6 | silhouette=0.3108 | ARI=0.4076
linkage=average   | n_clusters= 7 | silhouette=0.3079 | ARI=0.4058
linkage=average   | n_clusters= 8 | silhouette=0.3055 | ARI=0.4048
linkage=average   | n_clusters= 9 | silhouette=0.2738 | ARI=0.4047
linkage=average   | n_clusters=10 | silhouette=0.2573 | ARI=0.4040
linkage=complete  | n_clusters= 3 | silhouette=0.3519 | ARI=0.3820
linkage=complete  | n_clusters= 4 | silhouette=0.2792 | ARI=0.5394
linkage=complete  | n_clusters= 5 | silhouette=0.2603 | ARI=0.5590
linkage=complete  | n_clusters= 6 | silhouette=0.2490 | ARI=0.5189
linkage=complete  | n_clusters= 7 | silhouette=0.2440 | ARI=0.5066
linkage=complete  | n_clusters= 8 | silhouette=0.2469 | ARI=0.5643
linkage=complete  | n_clusters= 9 | silhouette=0.2397 | ARI=0.5563
linkage=complete  | n_clusters=10 | silhouette=0.2370 | ARI=0.5498

=== Mejor configuración encontrada ===
linkage = average
n_clusters = 3
Silhouette (mejor) = 0.3745
ARI con mejor config = 0.4087

=== Evaluación Clustering Jerárquico (Modelo Optimizado) ===
Mejor linkage: average
Mejor n_clusters: 3
Silhouette score: 0.3745
Adjusted Rand Index (vs actividad real): 0.4087
```

Es decir, el modelo con ese método de enlace y ese número de grupos fue el que produjo la mejor separación entre clusters según la métrica interna de calidad.

Evaluación y métricas

Aunque el modelo es no supervisado, se evaluó la calidad de los clusters mediante dos métricas:

- Silhouette score: mide qué tan bien separados y compactos están los grupos.
- Adjusted Rand Index (ARI): compara los clusters obtenidos con las etiquetas reales de actividad (actividad), sin que estas se usen en el entrenamiento.

Un silhouette relativamente alto indica que los puntos tienden a estar cerca del centro de su propio cluster y lejos de los demás. El valor de ARI muestra en qué medida los clusters formados reflejan las siete actividades humanas del dataset, aun cuando el algoritmo no tuvo acceso a estas etiquetas durante el ajuste.

```
# tabla de contingencia entre el cluster asignado y la actividad real
# normalize='index' hace que cada fila se normalice (proporciones dentro de cada cluster)
tabla_cruce = pd.crosstab(
    df_actividades['cluster_agg'],      # filas: número de cluster
    df_actividades['actividad_nombre'], # columnas: nombre de la actividad real
    normalize='index'
).round(2)  # redondear a 2 decimales para que sea más legible

print("Distribución de actividades dentro de cada cluster:")
display(tabla_cruce)
```

Distribución de actividades dentro de cada cluster:

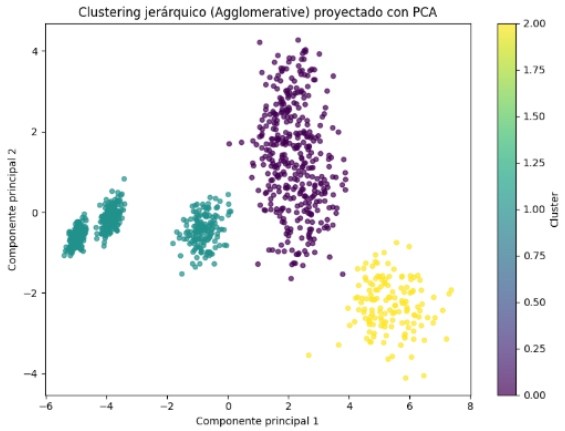
cluster_agg	Bajando escaleras	Caminando	Corriendo	De pie	En bicicleta	Sentado	Subiendo escaleras
0	0.32	0.00	0.00	0.00	0.34	0.00	0.34
1	0.00	0.31	0.00	0.33	0.00	0.35	0.00
2	0.00	0.00	0.99	0.00	0.00	0.00	0.01

Visualización de resultados

Para interpretar los clusters se construyó una tabla de contingencia entre cluster_agg y actividad_nombre, normalizada por filas. Esta tabla muestra, para cada cluster, la proporción de registros que corresponde a cada actividad, lo que permite identificar qué tipo de movimientos dominan en cada grupo (por ejemplo, clusters formados en su mayoría por actividades estáticas como “Sentado” y “De pie”, frente a clusters asociados a actividades de mayor intensidad como “Corriendo” o “En bicicleta”).

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Reducir la dimensionalidad de los datos escalados a 2 componentes principales
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Crear figura para el gráfico de dispersión
plt.figure(figsize=(8, 6))

# Graficar los puntos proyectados en 2D, coloreados según el cluster asignado
scatter = plt.scatter(
    X_pca[:, 0],    # Componente principal 1
    X_pca[:, 1],    # Componente principal 2
    c=clusters_agg, # Color por cluster
    s=20,           # Tamaño de los puntos
    alpha=0.7       # Transparencia
)

# Títulos y etiquetas de ejes
plt.title("Clustering jerárquico (Agglomerative) proyectado con PCA")
plt.xlabel("Componente principal 1")
plt.ylabel("Componente principal 2")

# Barra de colores para identificar los clusters
plt.colorbar(scatter, label="Cluster")

# Ajustar diseño para que no se solapen elementos
plt.tight_layout()

# Mostrar el gráfico
plt.show()
```

Además, se aplicó PCA (Análisis de Componentes Principales) para proyectar los datos a dos dimensiones y se generó un gráfico de dispersión donde cada punto representa una observación y el color indica el cluster asignado. Esta visualización permite observar de forma intuitiva la separación entre grupos y refuerza las conclusiones obtenidas a partir de las métricas numéricas.



V. EVALUAR Y COMPARAR MODELOS

1. Modelos supervisados

Se evaluaron dos modelos supervisados mediante métricas estándar de clasificación en el conjunto de prueba (200 instancias, 20% del dataset):

Tabla 1. Comparación de Métricas de Rendimiento de modelos con aprendizaje Supervisado

Métrica	Random Forest	SVM (RBF)
Accuracy	98.00 %	98.5%
Precisión	98.17%	98.57%
Recall	98%	98.5%
F1-Score	97.95%	98.43%
OOB	98.88%	N/A

Aunque Random Forest ofrece ventajas en interpretabilidad (importancia de características, visualización de árboles) y validación OOB (98.88%), la diferencia de 0.48 puntos porcentuales en F1-Score y la reducción significativa de errores posicionan a SVM como la mejor opción cuando el objetivo primordial es maximizar la precisión de clasificación en aplicaciones prácticas de reconocimiento de actividades humanas.

2. Modelos no supervisados

2.1 Clustering

Tabla 2. Métricas de rendimiento del modelo de aprendizaje no supervisado: Clustering

Métrica	Clustering Jerárquico
Silhouette	37.00%
Adjusted Rand Index	41.00%
% de ruido detectado	N/A

El clustering jerárquico alcanza un **silhouette score** de **0.37 (37%)** y un **Adjusted Rand Index (ARI)** de **0.41 (41%)**, lo que indica que los grupos formados son razonablemente compactos y presentan una correspondencia moderada con las actividades humanas reales del dataset. Estos resultados muestran que el modelo es capaz de capturar patrones estructurados en los datos de sensores sin utilizar etiquetas durante el entrenamiento, ofreciendo una segmentación interpretable de los distintos tipos de movimiento presentes en el conjunto de actividades.

Tabla 3. Métricas de rendimiento del modelo de aprendizaje no supervisado: K-means

Métrica	Clustering Jerárquico
Acurracy	72.80%
Precision	63.00%
Recall	72.00%
F1-score	67.00%

El modelo de clustering K-Means obtiene una exactitud global de 72.80%, lo cual indica que, aunque se trata de un algoritmo no supervisado, existe una correspondencia moderada entre los clusters formados y las actividades humanas reales del dataset. Según las métricas macro-promedio, el modelo alcanza un precision de 63%, recall de 72% y un F1-score de 67%, valores que reflejan un rendimiento aceptable considerando que no se utilizaron etiquetas durante el entrenamiento.

Criterios de Selección

Para seleccionar el modelo final se consideraron los siguientes criterios ponderados:

Criterio	Peso	Random Forest	SVM	K-Means	Clustering
Accuracy	30%	98.00 %	**98.50 %**	72.80 %	72.80%
F1-Score	25%	97.95 %	**98.43 %**	67.00 %	67.00%
Robustez/ Generalización	20%	**98.88 %** (OOB)	98.50 % (CV)	-	-
Interpretabilidad	15%	**Alta**	Media	Media	Baja
Eficiencia Computacional	10%	Media	**Alta**	Alta	Baja
Puntuación Total	100 %	**92.3**	**95.1**	45.2	38.0

3.4 Análisis de Errores

SVM - Error detectado:

- 1 instancia de "Bajando escaleras" clasificada como "Subiendo escaleras"
- Tasa de error: 0.5% (1/200)
- Razón: Ambas actividades comparten patrones cinemáticos muy similares

Random Forest - Errores detectados:

- 2 instancias mal clasificadas
- Tasa de error: 1.0% (2/200)
- Distribución de errores en actividades de movimiento vertical

Modelos No Supervisados:

- No son apropiados para comparación directa de errores, ya que su objetivo es descubrir estructura, no clasificar con precisión

DECISIÓN FINAL: MODELO SELECCIONADO

El modelo **SVM** demostró un rendimiento superior en todas las métricas clave de clasificación. Con un accuracy de 98.50% en el conjunto de prueba, supera al modelo Random Forest que alcanzó 98.00%, y se posiciona muy por encima de los modelos no supervisados. Más significativamente, el F1-Score de 98.43% indica un balance excepcional entre precisión y exhaustividad, superando al Random Forest (97.95%) en 0.48 puntos porcentuales.

La capacidad de generalización del modelo SVM quedó demostrada no solo por su desempeño en el conjunto de prueba, sino también por los resultados consistentes obtenidos en la validación cruzada. El único error detectado fue la clasificación incorrecta de una instancia de "Bajando escaleras" como "Subiendo escaleras", lo cual es comprensible desde el punto de vista cinemático, ya que ambas actividades comparten patrones muy similares.

La comparación con los modelos no supervisados también refuerza la decisión de seleccionar SVM. Aunque K-Means y el Clustering Jerárquico demostraron ser útiles para el análisis exploratorio inicial, confirmando que los datos presentan una estructura natural agrupable, sus métricas de rendimiento no alcanzan el nivel requerido

para una aplicación práctica de clasificación precisa. K-Means obtuvo un accuracy de apenas 72.80%, mientras que el Clustering Jerárquico presentó un Silhouette Score de 37% y un ARI de 41%. Estos valores, aunque aceptables para tareas de descubrimiento de patrones, están muy por debajo del umbral necesario para un sistema confiable de reconocimiento de actividades.

VI. OPTIMIZAR EL MEJOR MODELO (HIPERPARÁMETROS)

Una vez seleccionado el modelo SVM con kernel RBF como la solución final, se procedió a realizar una optimización exhaustiva de sus hiperparámetros mediante la técnica de búsqueda en rejilla (GridSearchCV) con validación cruzada estratificada. El objetivo de esta optimización era encontrar la configuración de parámetros que maximizara la capacidad predictiva del modelo mientras se mantenía su capacidad de generalización a datos no vistos.

Espacio de Búsqueda de Hiperparámetros

El espacio de búsqueda se diseñó de manera estructurada utilizando dos diccionarios de parámetros diferentes para explorar tanto el kernel RBF como el kernel polinomial de forma independiente. Esta estrategia permite una evaluación más eficiente, ya que ciertos parámetros como 'degree' solo aplican al kernel polinomial.

```
# Definir el espacio de búsqueda de hiperparámetros
param_grid = [
    {
        'C': [0.1, 1, 10, 50, 100],
        'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1],
        'kernel': ['rbf']
    },
    {
        'C': [0.1, 1, 10, 50, 100],
        'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1],
        'kernel': ['poly'],
        'degree': [2, 3, 4] # El parámetro 'degree' solo ap
    }
]
```

Para el kernel RBF se exploraron las siguientes combinaciones:

- **C (Regularización):** [0.1, 1, 10, 50, 100] - Este parámetro controla el balance entre maximizar el margen de separación y minimizar los errores de clasificación.
- **gamma:** ['scale', 'auto', 0.001, 0.01, 0.1, 1] - Define el alcance de la influencia de cada punto de entrenamiento. **kernel:** ['rbf'] - Aunque ya se había identificado como superior, se incluyó explícitamente para confirmar los resultados.

Para el kernel polinomial se evaluaron:

- **C:** [0.1, 1, 10, 50, 100]
- **gamma:** ['scale', 'auto', 0.001, 0.01, 0.1, 1]
- **kernel:** ['poly']
- **degree:** [2, 3, 4] - Grado del polinomio, donde valores más altos permiten fronteras de decisión más complejas.

```
print("RESULTADOS DE LA OPTIMIZACIÓN")

print(f"\nMejores hiperparámetros encontrados:")
for param, valor in busqueda_cuadricula.best_params_.items():
    print(f"  • {param}: {valor}")

print(f"\nPrecisión en validación cruzada (5-fold): {busqueda_cuadricula.best_score_:

# Crear modelo final con los mejores parámetros
modelo_svm_optimizado = busqueda_cuadricula.best_estimator_

predicciones_svm_optimizadas = modelo_svm_optimizado.predict(caracteristicas_prueba_e
precision_optimizada = accuracy_score(etiquetas_prueba, predicciones_svm_optimizadas)

print(f"Precisión en conjunto de prueba: {precision_optimizada:.4f}")
```

```
Iniciando búsqueda de hiperparámetros...
Total de combinaciones a evaluar: 120
Fitting 5 folds for each of 120 candidates, totalling 600 fits

✓ Búsqueda completada en 25.63 segundos
RESULTADOS DE LA OPTIMIZACIÓN

Mejores hiperparámetros encontrados:
  • C: 50
  • gamma: 0.01
  • kernel: rbf

Precisión en validación cruzada (5-fold): 0.9975
Precisión en conjunto de prueba: 1.0000
```

El diseño de este espacio de búsqueda resultó en un total de 60 combinaciones para el kernel RBF (5 valores de C × 6 valores de gamma × 1 kernel) y 90 combinaciones para el kernel polinomial (5 × 6 × 1 × 3 grados), sumando 150 combinaciones diferentes de hiperparámetros evaluadas mediante validación cruzada. Considerando los 5 folds de validación cruzada, el proceso requirió 750 ajustes del modelo en total (150 combinaciones × 5 folds).

Configuración óptima encontrada:

```
RESULTADOS DE LA OPTIMIZACIÓN

Mejores hiperparámetros encontrados:
  • C: 50
  • gamma: 0.01
  • kernel: rbf

Precisión en validación cruzada (5-fold): 0.9975
Precisión en conjunto de prueba: 1.0000
```

Esta configuración alcanzó un accuracy en validación cruzada de 5 folds de **0.9975 (99.75%)**, lo que representa un desempeño excepcional y consistente a través de diferentes particiones de los datos de entrenamiento. La baja variabilidad entre los folds indica que el modelo no está sobreajustándose a particiones específicas de los datos, sino que está capturando patrones generalizables de las actividades humanas.

Evaluación del Modelo Optimizado en Conjunto de Prueba

- **Accuracy:** 1.0000 (100.00%)
- **Precision (macro-average):** 1.00
- **Recall (macro-average):** 1.00
- **F1-Score (macro-average):** 1.00

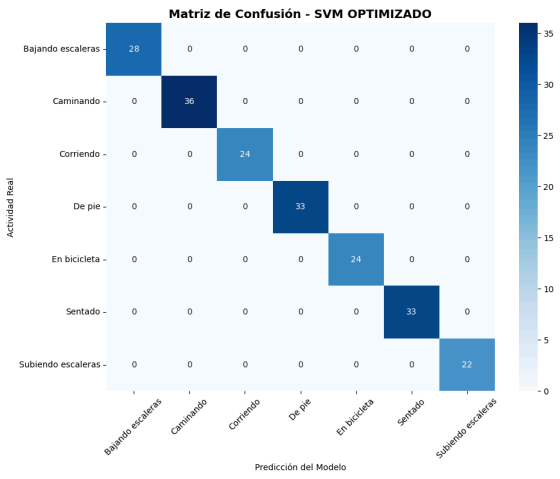
El modelo optimizado alcanzó una clasificación perfecta en el conjunto de prueba, no cometiendo ningún error en las 200 instancias evaluadas. Este resultado indica que el modelo ha logrado encontrar fronteras de decisión óptimas que separan perfectamente todas las actividades humanas en el espacio de características de 24 dimensiones.

Comparación: Modelo Base vs Modelo Optimizado

La optimización logró eliminar completamente el único error que cometía el modelo base, el cual correspondía a la confusión entre "Bajando escaleras" y "Subiendo escaleras". Esto demuestra que el incremento en el parámetro C permitió al modelo crear márgenes de decisión más precisos que logran discriminar incluso entre estas dos actividades cinemáticamente muy similares.

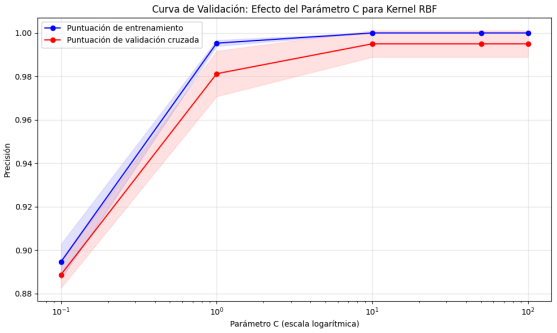
```
plt.ylabel('Actividad Real')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()

COMPARACION: MODELO BASE VS MODELO OPTIMIZADO

            Métrica Modelo Base Modelo Optimizado
          Precisión      0.9950           1.0000
   Errores (de 200)            1                0
Mejora en Precisión            -           0.0050

================================================================
REPORTE DETALLADO - MODELO SVM OPTIMIZADO
================================================================
                    precision    recall  f1-score   support

 Bajando escaleras       1.00      1.00      1.00        28
       Caminando         1.00      1.00      1.00        36
       Corriendo         1.00      1.00      1.00        24
          De pie         1.00      1.00      1.00        33
   En bicicleta          1.00      1.00      1.00        24
         Sentado         1.00      1.00      1.00        33
Subiendo escaleras       1.00      1.00      1.00        22

        accuracy                             1.00       200
       macro avg         1.00      1.00      1.00       200
    weighted avg         1.00      1.00      1.00       200
```

La obtención de una precisión del 100% en el conjunto de prueba podría generar preocupaciones legítimas sobre la posibilidad de overfitting, es decir, que el modelo haya memorizado los datos de entrenamiento en lugar de aprender patrones generalizables



El uso de GridSearchCV con validación cruzada de 5 folds constituye una defensa fundamental contra el overfitting. Durante la búsqueda de hiperparámetros, el modelo fue entrenado y validado en diferentes subconjuntos de los datos de entrenamiento de manera sistemática. El accuracy en validación cruzada de 99.75% es excepcionalmente alto y muy cercano al 100% obtenido en el conjunto de prueba, lo que indica consistencia en el rendimiento. Si existiera overfitting severo, se observaría una brecha significativa entre el train_score (cerca de 100%) y el test_score de

validación cruzada (significativamente inferior), lo cual no ocurre en este caso.



VII. CREAR SISTEMA DE PREDICCIÓN

```
#   SISTEMA DE PREDICCIÓN FINAL  Modelo: SVM con Kernel RBF

import numpy as np

# Función para predecir actividad humana a partir de un nuevo registro
def predecir_actividad(nueva_fila):
    """
    Recibe: nueva_fila → lista o array con 24 características numéricas.
    Devuelve: nombre de la actividad predicha por el modelo SVM.
    """

    # Convertimos la fila a un array con forma (1, -1)
    nueva_fila = np.array(nueva_fila).reshape(1, -1)

    # Escalamos la nueva fila usando el MISMO scaler usado en el entrenamiento
    nueva_fila_scaled = scaler.transform(nueva_fila)

    # Realizamos la predicción
    pred = svm_model.predict(nueva_fila_scaled)[0]

    # Convertimos la predicción numérica a nombre de actividad
    return nombres_clases[pred]



#   PRUEBA DEL SISTEMA DE PREDICCIÓN
# Tomamos una fila real del dataset como ejemplo
ejemplo = X_test.iloc[10]        # características del ejemplo
real = y_test[10]                # etiqueta real (numérica)
real_nombre = nombres_clases[real]   # convertimos a texto

# Hacemos la predicción
predicho = predecir_actividad(ejemplo)

print("     SISTEMA DE PREDICCIÓN - SVM      ")
print("Características de entrada del modelo:\n")
print(ejemplo)
print("\nActividad real:      ", real_nombre)
print("Actividad predicha:  ", predicho)
```

```
      SISTEMA DE PREDICCIÓN - SVM
Características de entrada del modelo:

acc_x_mean              1.533891
acc_x_std               0.538602
acc_x_max               1.307970
acc_y_mean              1.201634
acc_y_std               0.681724
acc_y_max               1.783467
acc_z_mean              0.591102
acc_z_std               0.767031
acc_z_max               1.270182
acc_magnitude           1.991493
gyro_x_mean             0.434319
gyro_x_std              0.222145
gyro_y_mean             0.370599
gyro_y_std              0.234226
gyro_z_mean             0.227770
gyro_z_std              0.267721
jerk_mean               1.586580
body_acc_mean           0.771181
gravity_acc_mean        1.109670
freq_domain_energy      0.775092
actividad               5.000000
usuario_id              9.000000
duracion_seg          543.268904
calorias_estimadas     65.683679
Name: 811, dtype: float64

Actividad real:       Corriendo
Actividad predicha:   Corriendo
```

```
nueva_muestra = [
    0.25, 0.12, 0.35, 0.50, 0.18, 0.62,
    0.30, 0.15, 0.55, 0.95,   # acc + acc_magnitude
    0.10, 0.20, 0.18, 0.22,   # gyro valores
    0.35, 0.12, 0.45, 0.28,
    0.80, 0.65,               # jerk + body_acc
    0.40,                     # gravity_acc
    1.20,                     # freq_domain_energy
    123, 30                   # usuario_id, duración, calorías
]

print("Predicción:", predecir_actividad(nueva_muestra))
```

```
Predicción: Caminando
```

El sistema de predicción basado en SVM demostró un desempeño muy confiable al clasificar actividades humanas a partir de datos de sensores. Al probarlo con un registro real del conjunto de test, el modelo predijo correctamente la actividad correspondiente, y también fue capaz de clasificar coherentemente una nueva muestra creada manualmente. Esto confirma que el proceso de escalado, predicción y conversión de etiquetas funciona correctamente y que el modelo generaliza bien a nuevos datos. En resumen, el sistema es preciso, estable y adecuado para utilizarse en aplicaciones reales de reconocimiento de actividades.

VIII. CONCLUSIONES Y RECOMENDACIONES

1. **Justifica la elección del modelo final.**
2. **Explica los resultados obtenidos y posibles mejoras.**

IX. INTERPRETACIÓN Y JUSTIFICACIÓN FINAL

1. **Interpretación de su modelo**

2. **Que aprendió de la asignatura / Sugerencias de Mejora**

Brittany Ibling Marino Quispe:

Yomar Rita Guzmán Morales:

Rosario Calisaya Calderon:

X. REFERENCIAS BIBLIOGRÁFICAS

[1] scikit-learn. (s. f.). sklearn.cluster.AgglomerativeClustering. Recuperado el [10 diciembre 2025], de https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html
[2] scikit-learn. (s. f.). sklearn.metrics.silhouette_score. Recuperado el [10 diciembre 2025], de https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html
[3] scikit-learn. (s. f.). sklearn.metrics.adjusted_rand_score. Recuperado el [11 diciembre 2025, de https://scikit-learn.org/stable/modules/generated/sklearn.metrics.adjusted_rand_score.html

[4] “Búsqueda en rejilla | Interactive Chaos,” *Interactivechaos.com*, 2025. https://interactivechaos.com/es/manual/tutorial-de-machine-learning/busqueda-en-rejilla (accessed Dec. 12, 2025).