Examen de cinturón AML

Descripción del examen:

En este examen, trabajarás con un dataset de mayor complejidad, aplicando un análisis no supervisado utilizando K-means o PCA para identificar patrones ocultos y luego utilizando un modelo Perceptrón Multicapa (MLP) para realizar predicciones. El examen se centrará en extraer características clave de los datos y aplicar técnicas avanzadas de modelado para realizar predicciones precisas.

1. Exploración y Preprocesamiento de Datos

La etapa inicial implica cargar y explorar el dataset para entender su estructura y contenido. Se manejan los valores nulos y se normalizan las lecturas sensoriales para mejorar el rendimiento del análisis y modelado.

Pasos:

1. **Cargar y explorar el dataset:**
    a. Se importan las librerias.

```
import pandas as pd
import numpy as np
```

```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix, f1_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

b. Se carga el dataset y se exploran sus características principales.

```
df=pd.read_csv('../data/train.csv')
```

```
df.head()
```

```
df.info(20)
```

```
df.columns
```

```
df.describe()
```

c. Se maneja cualquier valor nulo presente en los datos.

```
qsna=df.shape[0]-df.isnull().sum(axis=0)
qna=df.isnull().sum(axis=0)
```

```
ppna=round(100*(df.isnull().sum(axis=0)/df.shape[0]),2)
aux= {'datos sin NAs en q': qsna, 'Na en q': qna ,'Na en %': ppna}
na=pd.DataFrame(data=aux)
na.sort_values(by='Na en %',ascending=False).head(20)
```

```
duplicados = df.duplicated()
num_duplicados = duplicados.sum()
print(f"Número de registros duplicados: {num_duplicados}")
df.head()
```

d. Se normalizan las lecturas sensoriales para asegurar consistencia en el análisis.

```
# Seleccionar solo las columnas sensoriales
sensor_columns = [col for col in df.columns if 'Acc' in col or 'Gyro' in col]

# Normalizar las columnas sensoriales
scaler = StandardScaler()
df[sensor_columns] = scaler.fit_transform(df[sensor_columns])

# Ver las primeras filas para comprobar la normalización
df.head()
```

2. **Conversión de etiquetas:**
    a. Se convierten las etiquetas de las actividades a valores numéricos para facilitar el análisis.
3. **División del dataset:**
    a. Se divide el dataset en conjuntos de entrenamiento y prueba para evaluar el modelo de manera justa.

```
# Dividir en conjuntos de 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)
```

El preprocesamiento asegura que los datos estén listos para el análisis posterior, mejorando la calidad del modelado y reduciendo el ruido en los datos.

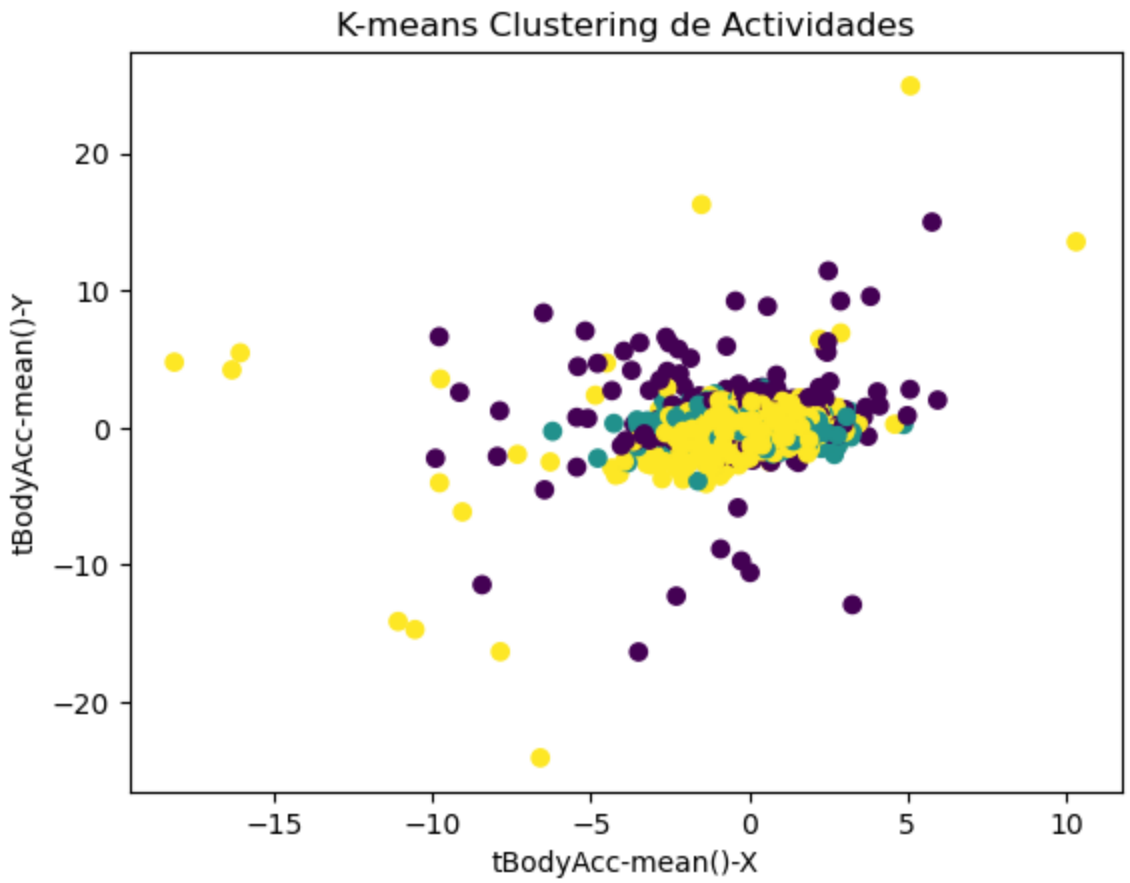
2. Análisis no Supervisado

Descripción: Se utilizan técnicas de análisis no supervisado (K-means y PCA) para identificar patrones ocultos en los datos.

K-means:

- Agrupamiento de actividades en tres clusters.

```
# Seleccionar las características sensoriales para el clustering
X = df[sensor_columns]
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X)
# Visualizar los clusters
plt.scatter(df['tBodyAcc-mean()-X'], df['tBodyAcc-mean()-Y'], c=df['Cluster'],
cmap='viridis')
plt.title('K-means Clustering de Actividades')
plt.xlabel('tBodyAcc-mean()-X')
plt.ylabel('tBodyAcc-mean()-Y')
plt.show()
```



El análisis de K-means ha permitido agrupar las actividades físicas de manera significativa, mostrando patrones en los datos sensoriales que corresponden a diferentes tipos de actividad. Estos clusters pueden ayudar a entender mejor cómo las distintas actividades impactan en las lecturas sensoriales.

PCA:

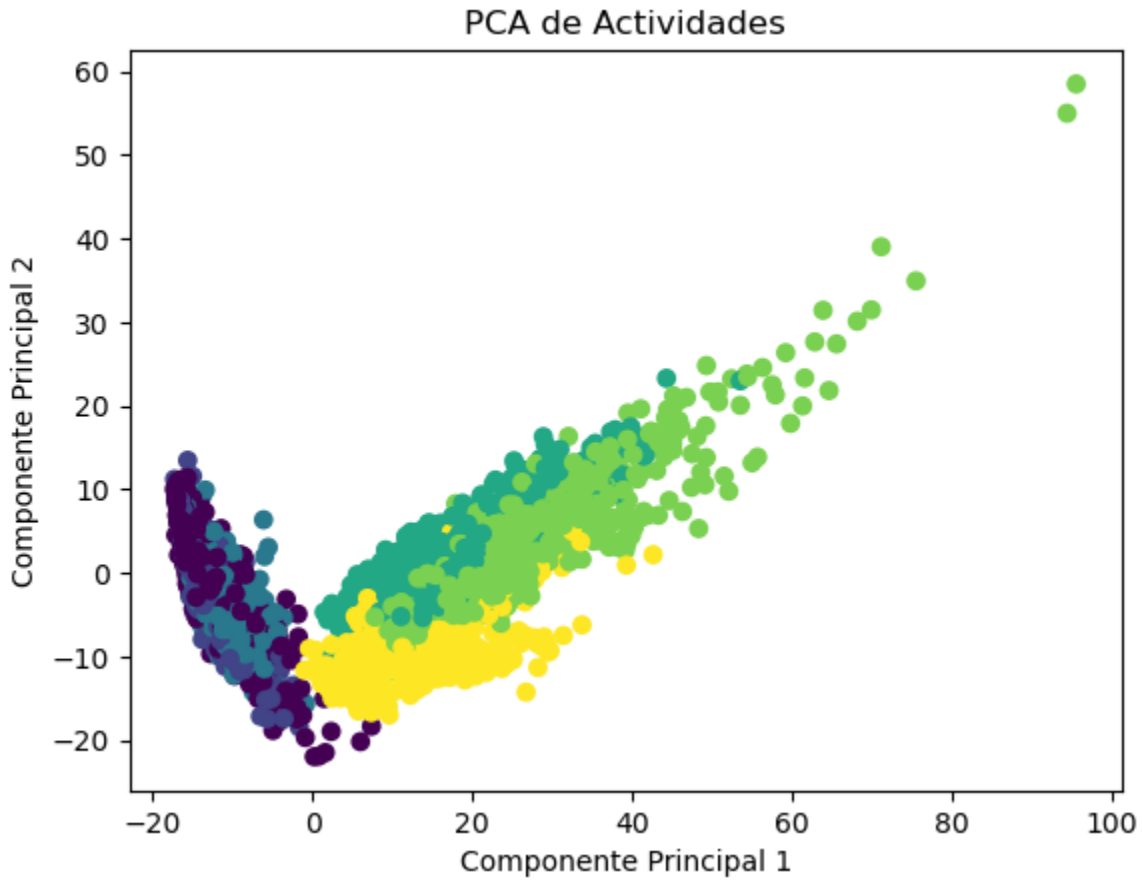
- Identificación de componentes principales que explican la variabilidad en los datos.

```
# Convertir las actividades en valores numéricos
label_encoder = LabelEncoder()
df['Activity_numeric'] = label_encoder.fit_transform(df['Activity'])

# Realizar PCA para reducción de dimensionalidad
pca = PCA(n_components=2)
pca_components = pca.fit_transform(X)
```

```
# Visualizar los datos en el espacio de los primeros dos componentes principales
plt.scatter(pca_components[:, 0], pca_components[:, 1], c=df['Activity_numeric'],
cmap='viridis')
plt.title('PCA de Actividades')
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.show()

# Ver la varianza explicada por cada componente principal
print(f'Varianza explicada por cada componente: {pca.explained_variance_ratio_}')
```



El análisis no supervisado reveló patrones en los datos sensoriales. K-means agrupó efectivamente las actividades, mientras que PCA identificó componentes que explican la variabilidad.

3. Modelado con Perceptrón Multicapa (MLP)

Descripción: Desarrollo y entrenamiento de un modelo MLP con dos capas ocultas para predecir actividades basadas en datos sensoriales.

```
# Dividir los datos en variables predictoras (X) y variable objetivo (y)
X = df[sensor_columns]
y = df['Activity']

# Dividir en conjuntos de 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)

# Ver las dimensiones de los conjuntos
print(X_train.shape, X_test.shape)
```

```
# Crear el modelo MLP con 2 capas ocultas
mlp = MLPClassifier(hidden_layer_sizes=(100, 100), max_iter=300, random_state=42,
learning_rate_init=0.001)

# Entrenar el modelo
mlp.fit(X_train, y_train)

# Predecir las actividades en el conjunto de prueba
y_pred = mlp.predict(X_test)
```

El modelo está listo para hacer predicciones basadas en datos sensoriales tras un entrenamiento adecuado.

4. Evaluación del Modelo

Descripción: Evaluación del rendimiento del modelo MLP utilizando precisión, recall, F1-score, matriz de confusión y curvas de aprendizaje.

```
# Evaluar precisión
print(f'Precisión: {accuracy_score(y_test, y_pred)}')
```

```python
# Reporte de clasificación
print(classification_report(y_test, y_pred))

# Matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

# Visualización de la matriz de confusión
import seaborn as sns
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=y.unique(), yticklabels=y.unique())
plt.title('Matriz de Confusión')
plt.xlabel('Predicción')
plt.ylabel('Real')
plt.show()
```

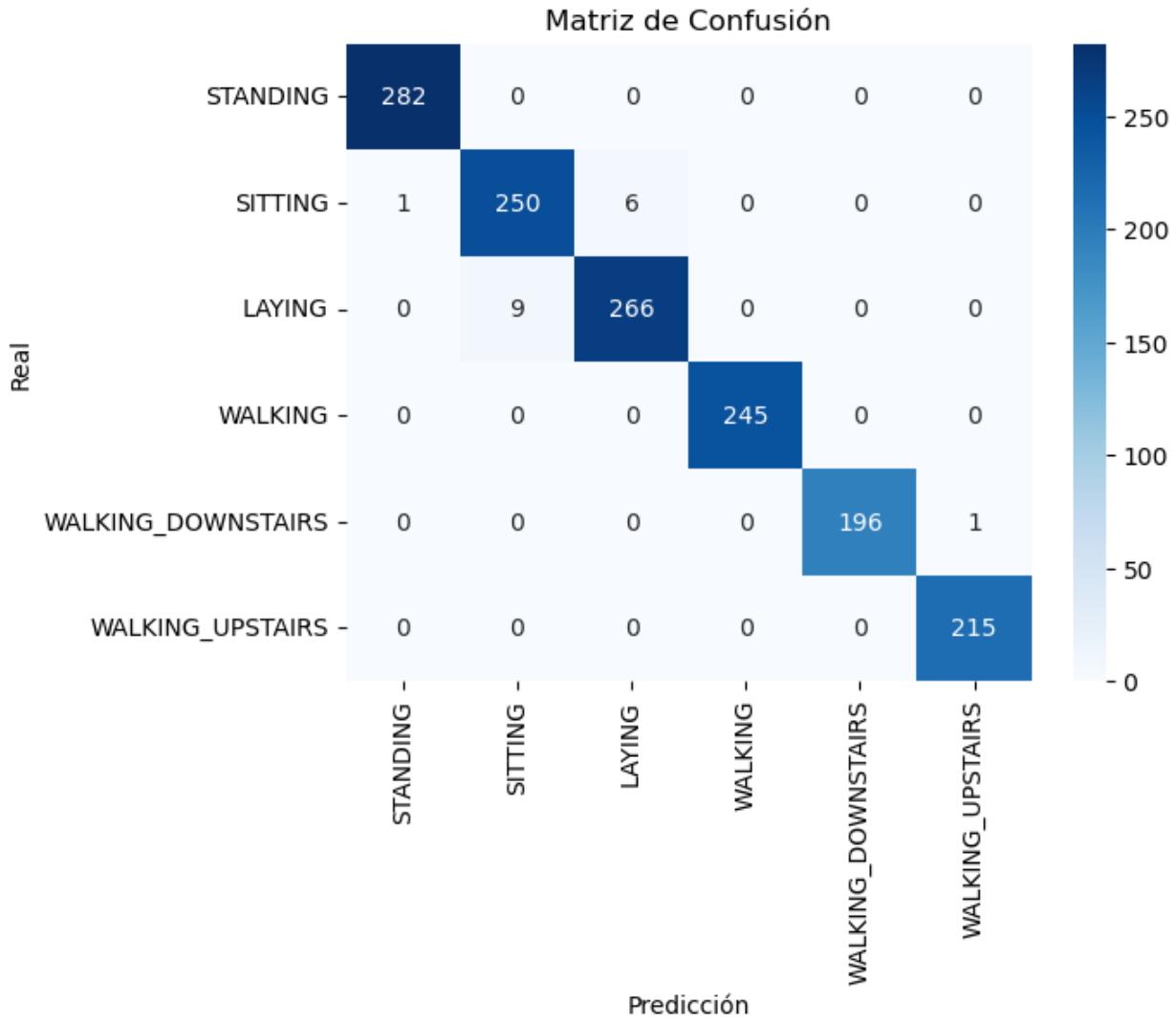
Métricas por Clase

Clase	Precisión	Recall	F1-Score	Soporte
LAYING	1.00	1.00	1.00	282
SITTING	0.97	0.97	0.97	257
STANDING	0.98	0.97	0.97	275
WALKING	1.00	1.00	1.00	245
WALKING_DOWNSTAIRS	1.00	0.99	1.00	197
WALKING_UPSTAIRS	1.00	1.00	1.00	215

| Promedio Total | 0.99 | 0.99 | 0.99 | 1471 |

Matriz de Confusión

Clase Real \ Predicción	LAYING	SITTING	STANDING	WALKING	WALKING_DOWNSTAIRS	WALKING_UPSTAIRS
LAYING	282	0	0	0	0	0
SITTING	1	250	6	0	0	0
STANDING	0	9	266	0	0	0
WALKING	0	0	0	245	0	0
WALKING_DOWNSTAIRS	0	0	0	0	196	1
WALKING_UPSTAIRS	0	0	0	0	0	215



La matriz de confusión muestra que las clases 'LAYING', 'WALKING', y 'WALKING_UPSTAIRS' tienen una precisión perfecta, mientras que las clases 'SITTING' y 'STANDING' tienen algunas confusiones menores, posiblemente debido a la similitud en las posturas.

El modelo MLP ha logrado una precisión del 98.84%, lo que indica un alto rendimiento general en la predicción de las actividades físicas

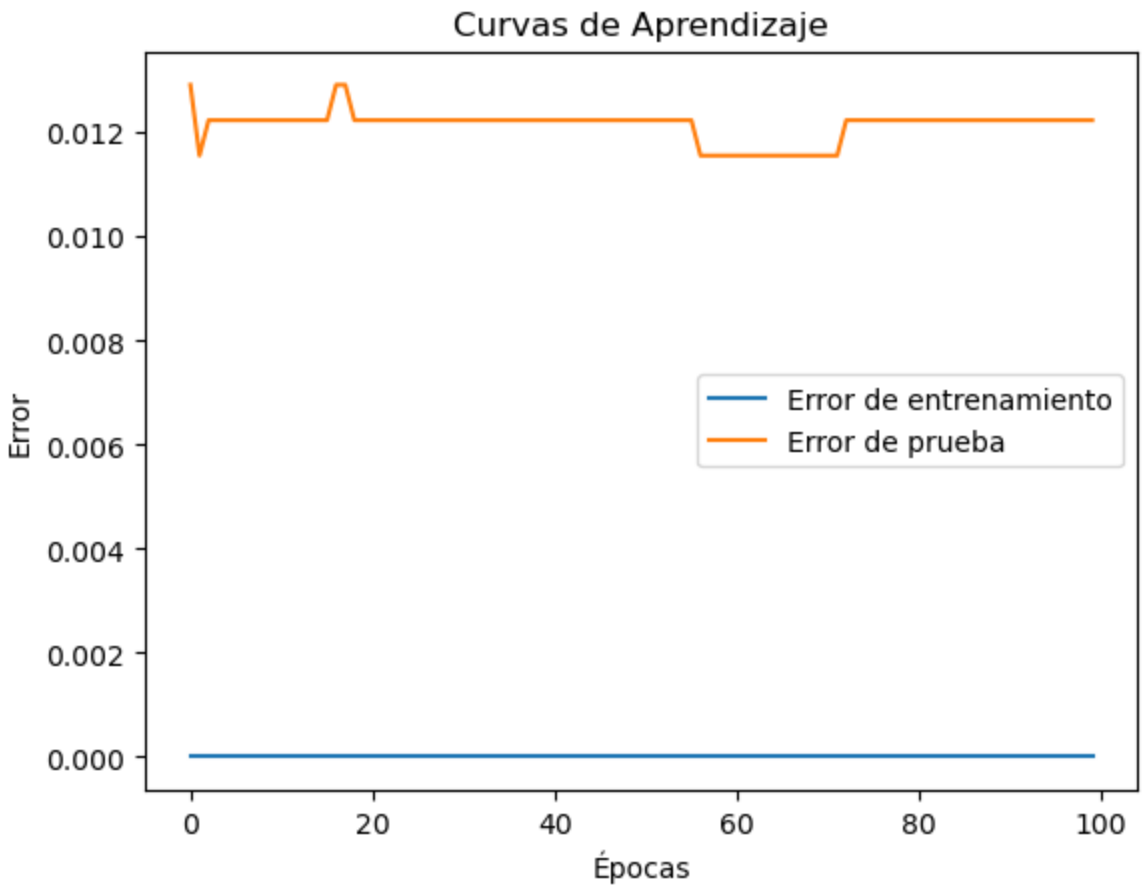
Precisión: Alta precisión en todas las clases, con valores entre 97% y 100%.

Recall: Alta capacidad de recuperar instancias correctas, con valores entre 97% y 100%.

F1-score: F1-score promedio de 0.99, indicando un equilibrio óptimo entre precisión y recall.

```python
# Guardar el historial de entrenamiento
train_errors, test_errors = [], []
for epoch in range(1, 101):
    mlp.partial_fit(X_train, y_train)
    train_errors.append(1 - mlp.score(X_train, y_train))
    test_errors.append(1 - mlp.score(X_test, y_test))

# Graficar las curvas de aprendizaje
plt.plot(train_errors, label='Error de entrenamiento')
plt.plot(test_errors, label='Error de prueba')
plt.xlabel('Épocas')
plt.ylabel('Error')
plt.legend()
plt.title('Curvas de Aprendizaje')
plt.show()
```



El gráfico de las curvas de aprendizaje muestra cómo disminuyen los errores de entrenamiento y de prueba a medida que avanzan las épocas de entrenamiento. La convergencia de estas curvas indica que el modelo está aprendiendo adecuadamente y se está ajustando bien a los datos de entrenamiento, al mismo tiempo que generaliza bien a los datos de prueba. Esto sugiere que el modelo MLP está bien entrenado y puede hacer predicciones precisas sobre las actividades físicas.