Clasificación de actividades por bases de datos de articulaciones 3D

Frias Cortez Rafael Alejandro. Ingeniería de Datos e Inteligencia Artificial

Resumen – En esta práctica se trabajó con datos de 25 articulaciones humanas capturadas por un Kinect durante tres actividades (taladrar, martillar y atornillar) realizadas por seis personas. Se extrajeron 8 distancias entre las articulaciones y se calcularon métricas como promedio, varianza, mínimo, máximo y velocidad para cada actividad. Estos datos fueron utilizados para entrenar y evaluar varios clasificadores (Adaboost, KNN-Minkowski, KNN-Cosine, LDA, Fifth-SVM), midiendo su desempeño mediante métricas como exactitud, precisión, recall y F1-score. Se implementaron filtros para mejorar la calidad de los datos.

I. INTRODUCCIÓN

La detección de articulaciones humanas en 3D es importante para actividades en la interacción humano-computadora. Los sistemas de motion capture ofrecen alta precisión, los sistemas RGB-D, como el Kinect, proporcionan una alternativa económica y suficientemente precisa para tareas de localización y reconocimiento de actividades. En esta práctica, se trabajó con datos de posiciones 3D de 25 articulaciones del cuerpo humano para modelar y clasificar actividades básicas de la industria metal-mecánica (taladrar, martillar y atornillar), utilizando técnicas de clasificación y análisis de secuencias de movimientos.

II. METODOS

Para cada muestra de actividad, se calculó un conjunto de características a partir de **8 distancias** entre articulaciones del cuerpo humano (codo, muñeca, rodilla, talones.. Estas distancias fueron referenciadas al punto central del cuerpo (joint 0). Además, se calcularon las **velocidades** entre frames.

Para cada distancia y velocidad, se extrajeron las siguientes estadísticas:

- Promedio:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

- Varianza:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

- Minimo y maximo:

$$\min(x), \max(x)$$

Clasificadores Utilizados [1]

1. AdaBoost

- Es un método que combina varios modelos simples (llamados "débiles") para hacer uno más fuerte.
- Si un modelo comete errores, el siguiente intenta corregirlos.
- Al final, todos los modelos votan para decidir la clase correcta.

2. KNN (K-Nearest Neighbors)

- Este clasificador se basa en la idea de que "las cosas similares están cerca".
- Cuando llega un nuevo ejemplo, busca los **K ejemplos más cercanos** y vota por la clase más común entre ellos.
    - **Minkowski**: mide la distancia como si contaras los pasos en línea recta o por cuadras (según el valor de p).
    - **Coseno**: compara el **ángulo** entre dos vectores, no la distancia directa.

3. LDA (Linear Discriminant Analysis)

- Busca una línea (o plano) que separe lo mejor posible las clases.
- Proyecta los datos para que las clases estén lo más alejadas entre sí.
- Funciona bien cuando los datos tienen forma de "nube" y cada clase está agrupada.

4. Fifth-SVM

- Es una versión del algoritmo SVM que busca separar las clases con una curva (no con una línea recta).
- Usa una función matemática (un polinomio de grado 5) que permite encontrar separaciones más complejas entre los datos.

Evaluación

Para cada clasificador se generaron:

- **Matriz de confusión**
  Muestra los aciertos y errores por clase.
- **Métricas de desempeño**
  - **Exactitud (Accuracy)**
  - **Precisión**
  - **Recall (Sensibilidad)**
  - **F1-Score**

Estas métricas permitieron comparar objetivamente el rendimiento de los clasificadores y seleccionar el más adecuado según la actividad a identificar.

$$precision = \frac{TP}{TP + FP}$$
$$recall = \frac{TP}{TP + FN}$$
$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$
$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

Imagen 1. Fórmulas de las métricas de desempeño.

Una **matriz de confusión** es una herramienta para evaluar el rendimiento de un clasificador, especialmente cuando estamos tratando con múltiples clases. Para una **matriz de confusión de 3x3**, cada fila representa una **clase real**, y cada columna representa una **clase predicha**. La idea principal de las matrices es comparar las **predicciones** del modelo con las **etiquetas reales** para cada clase.

En una **matriz de confusión 3x3**, como hay tres clases, tendrás tres filas y tres columnas, cada una correspondiente a las tres clases:

	Predicha 0	**Predicha 1**	**Predicha 2**
Real 0	TP_00	FP_01	FP_02
Real 1	FN_10	TP_11	FP_12
Real 2	FN_20	FN_21	TP_22

Tabla 1. Explicación de matrices de confusión.

Elementos de la matriz de confusión

- **TP (True Positive)**: Son los casos en los que **el modelo predijo correctamente** la clase. En la matriz, esto se encuentra en la diagonal principal,
  - En **TP_00** (donde el valor real y el predicho es 0), representa el número de veces que el modelo predijo correctamente la clase 0.
- **FP (False Positive)**: Son los casos en los que el modelo **predijo incorrectamente** una clase, asignando una etiqueta errónea cuando la clase real era diferente.
  - En **FP_01** (donde la clase real era 0, pero el modelo predijo 1), es el número de veces que la clase 0 fue confundida con la clase 1.
- **FN (False Negative)**: Son los casos en los que el modelo **no logró predecir correctamente** una clase real, predijo una clase incorrecta.
  - En **FN_10** (donde la clase real era 1, pero el modelo predijo 0), representa cuántas veces el modelo no predijo correctamente la clase 1.
- **TN (True Negative)**: No aparece explícitamente en la matriz, pero se puede calcular como **todos los otros valores que no están en la fila y columna de la clase en cuestión**. Los verdaderos negativos son todas las instancias que **no pertenecen a la clase actual**, y fueron correctamente predichas como no pertenecientes a esa clase.

III. RESULTADOS

Para comenzar con la práctica, se realizó la lectura de los archivos .txt correspondientes a cada acción realizada por cada persona. Estos archivos contenían las posiciones de los **joints** (articulaciones) capturados en cada frame.
La lectura se organizó utilizando rutas hacia las carpetas correspondientes y se guio mediante un archivo .csv que indicaba los **fragmentos relevantes** (frame de inicio y de fin) para cada acción. Se extrajeron los puntos necesarios de cada secuencia de manera estructurada.
Se implementó una visualización en formato **2D** del esqueleto humano a partir de los puntos extraídos, con el fin de observar el movimiento realizado en cada acción. Para construir estas visualizaciones, se definieron conexiones entre los joints clave, tales como:

- Columna ↔ Cabeza
- Hombros ↔ Codos ↔ Muñecas
- Cadera ↔ Rodillas ↔ Talones

Estas uniones se aplicaron para ambos lados del cuerpo (izquierdo y derecho), permitiendo representar de forma clara la postura y el movimiento del sujeto en cada actividad.
Antes de calcular características como distancias y velocidades, se aplicó un **suavizado por promedio móvil** a las posiciones de las articulaciones (joints) para reducir el ruido en los datos capturados por el sensor.

Se implementa una función suavizar_joints, que recorre todos los frames de la secuencia y reemplaza la posición de cada frame por el **promedio de sus vecinos cercanos**.
Esto ayuda a suavizar pequeños errores o fluctuaciones que puedan haber ocurrido en la captura del movimiento.

Funcionamiento:

- Para cada frame, se toma una **ventana** de tamaño definido (por ejemplo, 5 frames).
- Se promedian las posiciones de los joints dentro de esa ventana.
- El frame actual se reemplaza por ese promedio.
- El resultado es una secuencia más estable y continua, ideal para análisis de movimiento.

Ventajas:

- Reduce el **ruido de alta frecuencia** (saltos bruscos o errores de medición).
- Mejora la **precisión de características derivadas**, como velocidades y distancias.
- Es un método no invasivo que **preserva la forma general del movimiento**.

Ya con el calculo de las distancias y las velocidades de dichas distancias se hizo el cálculo de las características para posteriormente juntar todas las características de las acciones de cada persona, para poder hacer uso de los clasificadores.

Preparación de Datos para el Entrenamiento

Una vez obtenidas las características de cada muestra de actividad, se procedió a preparar los datos para el entrenamiento de los clasificadores. Este proceso incluyó los siguientes pasos:

1. **Carga del Conjunto de Datos**
   Se lee el archivo CSV caracteristicas_completo_filtro.csv, el cual contiene todas las muestras extraídas, con sus respectivas etiquetas (actividad realizada) y datos complementarios (persona y repetición).
2. **Selección de Variables**
   Se eliminaron las columnas no numéricas (persona, repeticion, actividad) para obtener solo las características cuantitativas que se utilizarán como entrada para los modelos. La columna actividad se conservó como etiqueta (y).
3. **Codificación de Etiquetas**
   Dado que los clasificadores trabajan con valores numéricos, las etiquetas categóricas correspondientes a las actividades fueron transformadas a valores enteros utilizando **Label Encoding**.
4. **Estandarización de Características**
   Para evitar que las diferencias de escala entre características afecten el rendimiento de los clasificadores, se aplicó una **normalización tipo Z-score** con StandardScaler, lo cual transforma cada característica a media 0 y desviación estándar 1.
5. **División de Datos**
   Se dividió el conjunto total en dos subconjuntos:
   - **70% para entrenamiento**
   - **30% para prueba (test)**

Esta división se realizó con la función train_test_split de sklearn, utilizando una **estratificación** basada en las etiquetas para garantizar que todas las clases estén representadas proporcionalmente en ambos subconjuntos.

Entrenamiento y Evaluación del Clasificador AdaBoost

Se entreno el primer modelo: **AdaBoost (Adaptive Boosting)**. Este método es un algoritmo de ensamble que combina múltiples clasificadores débiles (generalmente árboles de decisión simples) para formar un clasificador más robusto.

Implementación

Se utilizó la clase AdaBoostClassifier de la librería sklearn.ensemble, configurada con los siguientes parámetros:

- n_estimators=100: Se utilizaron 100 clasificadores débiles.
- random_state=42: Se fijó una semilla para asegurar reproducibilidad.

Evaluación

Se generaron las predicciones sobre el conjunto de prueba (y_pred_ada) y se evaluaron mediante el método classification_report, que proporciona métricas como:

- **Precisión** (Precision): Porcentaje de verdaderos positivos entre las predicciones positivas.
- **Exhaustividad** (Recall): Porcentaje de verdaderos positivos entre los casos reales positivos.
- **F1-Score**: Promedio armónico entre precisión y recall.
- **Accuracy**: Proporción de predicciones correctas totales.

Clasificador K-Nearest Neighbors (KNN) con Distancia de Minkowski

- Para optimizar el desempeño del clasificador **K-Nearest Neighbors (KNN)**, se implementó una búsqueda del mejor número de vecinos (k) utilizando validación cruzada. El proceso consistió en evaluar valores de k desde 1 hasta 20 con la métrica de distancia **Minkowski** y un parámetro p = 3, ¿Por qué p=3?
- Si p = 1 → distancia **Manhattan**
- Si p = 2 → distancia **Euclidiana**
- Si p = 3 → es una distancia **más sensible a las diferencias grandes** entre características (resalta más los valores que se alejan mucho)

Entonces, usar p=3 es una forma de **darle más peso a las diferencias grandes** entre los vectores de características. Puede ser útil si esperas que esas diferencias tengan un impacto importante en la clasificación.

La validación cruzada utilizada fue de **5 particiones (cv=5)**
¿Por qué cv=5?

- Balance entre precisión y tiempo de cómputo: Con 5 particiones, cada iteración usa el 80% de los datos para entrenar y el 20% para validar. Esto ofrece una estimación confiable del rendimiento del modelo sin tardar demasiado.
- Reducción del sesgo: Evaluar el modelo en diferentes subconjuntos del conjunto de entrenamiento ayuda a reducir el riesgo de que la elección del conjunto de validación influya excesivamente en el resultado.
- Es un valor estándar ampliamente aceptado (igual que cv=10), a menos que tengas muchos datos (entonces puedes usar más pliegues) o muy pocos (entonces usarías menos pliegues).

Lo que permite entrenar y validar el modelo con diferentes subconjuntos del conjunto de entrenamiento, obteniendo una estimación más robusta del rendimiento general. Para cada valor de k, se calculó la precisión promedio sobre las particiones, y se almacenaron los resultados para su posterior análisis.
Se identificó el valor de k que obtuvo la mayor precisión promedio, considerándolo como el valor óptimo para el clasificador. En este caso, el mejor resultado se obtuvo con
k = 9, lo que indica que el uso de 9 vecinos proporciona el equilibrio ideal entre sesgo y varianza para este problema de clasificación.
Adicionalmente, se generó una gráfica donde se muestra cómo varía la precisión promedio en función del número de vecinos, lo cual ayuda a visualizar la tendencia del modelo frente a cambios en este parámetro.

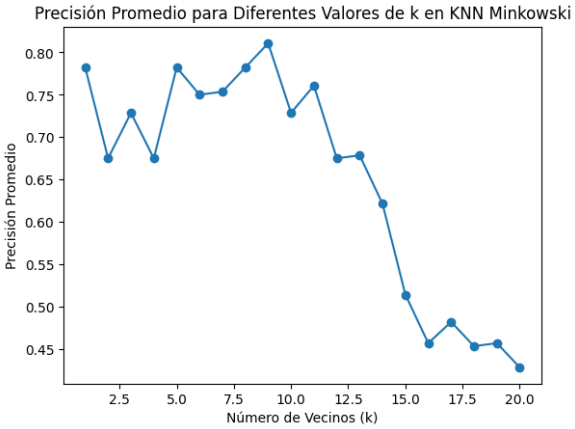

Imagen 2. Valores de K para KNN Minkowski

Clasificador K-Nearest Neighbors (KNN) con Distancia de Coseno

Rango de búsqueda:

- Se probaron valores de vecinos k desde 1 hasta 20.

Evaluación mediante validación cruzada:

- Para cada valor de k, se evaluó el modelo usando validación cruzada de 5 pliegues (cv=5). Esto significa que el conjunto de entrenamiento se dividió en 5 partes:
- El modelo se entrena en 4 y se valida en la restante, repitiendo este proceso 5 veces.
- Se calcula la precisión promedio para cada valor de k.

Selección del mejor k:

- Se identifica el k que obtuvo la mayor precisión promedio durante la validación cruzada.

Visualización:

- Se generó una gráfica de k vs. precisión promedio, permitiendo observar cómo evoluciona el desempeño del modelo al variar el número de vecinos.

Entrenamiento y evaluación final:

- Se entrenó un modelo definitivo con el mejor k usando la métrica de coseno, y se evaluó sobre el conjunto de prueba (X_test).

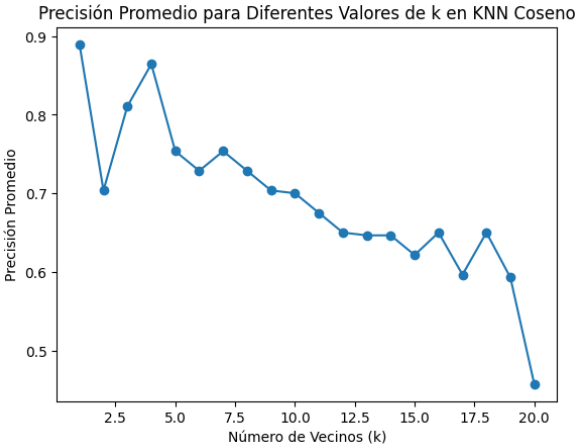

Imagen 3. Valores de K para KNN Coseno

Clasificador LDA (Linear Discriminant Analysis)

Para evaluar el desempeño del modelo **Linear Discriminant Analysis (LDA)** como clasificador, se utiliza la implementación de LinearDiscriminantAnalysis de la biblioteca scikit-learn. Aunque LDA es comúnmente conocido como una técnica de reducción de dimensionalidad, en esta práctica se aplicó como un **clasificador supervisado**.

- Se entrena el modelo lda con los datos de entrenamiento X_train y las etiquetas y_train.
- Luego, se utiliza el método predict() para obtener las predicciones sobre los datos de prueba.

El objetivo de LDA como clasificador es evaluar su capacidad para separar las clases presentes en el conjunto de datos, basándose en maximizar la separación lineal entre ellas. A diferencia de métodos como KNN, que pueden usar distancias, LDA genera una combinación lineal de características que maximiza la separación entre clases y minimiza la varianza dentro de cada clase.

Clasificador Fifth-SVM (SVM con Kernel Sigmoide)

Se empleó un **clasificador SVM (Máquinas de Vectores de Soporte)** utilizando un **kernel sigmoide** (kernel='sigmoid'), el cual en esta practica ha sido nombrado como *Fifth-SVM* para distinguirlo de otros clasificadores. El SVM es una técnica supervisada muy utilizada en problemas de clasificación, cuyo objetivo es encontrar el hiperplano que mejor separa las clases en el espacio de características.

- kernel='sigmoid': este tipo de kernel aplica una función sigmoide para transformar los datos, similar a una red neuronal, y es útil cuando la relación entre las clases no es lineal.
- C=1.0: es el parámetro de regularización, controla el equilibrio entre obtener un margen amplio y clasificar correctamente los puntos de entrenamiento.
- gamma='scale': controla el alcance de la influencia de cada punto de entrenamiento. 'scale' es un valor automático basado en la varianza de las características.

Resultados de los modelos:

Adaboost

	precision	recall	F1-score	support
Taladrar	0.80	0.80	0.80	5
Martillar	1.0	0.83	0.91	6
Atornillar	0.86	1.0	0.92	6
Accuracy			0.88	17
Macro avg	0.88	0.88	0.88	17
Weighted avg	0.88	0.88	0.88	17

Tabla 2. Resultados adaboost

Cada fila representa una clase diferente del conjunto de etiquetas (por ejemplo, tres actividades distintas).

- Precision:

Mide la proporción de verdaderos positivos entre todos los elementos predichos como positivos. Por ejemplo, para la clase 1, se predijo perfectamente: 100% de los elementos clasificados como clase 1 realmente lo eran.

- Recall:

Mide la proporción de verdaderos positivos capturados entre todos los reales. Por ejemplo, para la clase 2, el modelo capturó el 100% de los ejemplos verdaderos.

- F1- Score:

Es la media armónica entre la precisión y el recall. Representa el equilibrio entre ambos. Es útil cuando se quiere una métrica que penalice errores en ambos sentidos.

- Support:

Es la cantidad de muestras reales que hay de esa clase en el conjunto de prueba.

Clase 0(taladrar):

- Precision: 0.80 → 80% de las veces que se predijo clase 0, fue correcto.
- Recall: 0.80 → De todos los ejemplos reales de clase 0, el 80% fue correctamente identificado.
- F1-score: 0.80 → Equilibrio entre precisión y recall.
- Support: 5 → Había 5 ejemplos reales de la clase 0.

Métricas:

- **Accuracy (exactitud total):**
  0.88 → El modelo clasificó correctamente 88% del total de instancias (15 de 17 correctamente clasificadas).
- **Macro avg:**
  Promedio simple (sin ponderar) de precisión, recall y F1-score entre todas las clases:
  - Útil cuando las clases están desbalanceadas.
- **Weighted avg:**
  Promedio ponderado por el número de instancias en cada clase.
  - Más representativo del desempeño general si el conjunto está desbalanceado (ya que da más peso a las clases más frecuentes).

KNN Minkowski

	precision	recall	F1-score	support
Taladrar	0.60	0.60	0.60	5
Martillar	0.83	0.83	0.83	6
Atornillar	0.83	0.83	0.83	6
Accuracy			0.76	17
Macro avg	0.76	0.76	0.76	17
Weighted avg	0.76	0.76	0.76	17

Tabla 3. Resultados KNN Minkowski.

- Precision:

De las veces que el modelo predijo esa clase, señala cuántas fueron correctas

- Recall:

De las veces que realmente era esa clase, señala cuántas veces acertó el modelo.

- F1- Score:

Media armónica entre precisión y recall (mejor cuando ambas son altas)

- Support:

Cantidad real de muestras en esa clase en el conjunto de prueba.

Clase 0 (taladrar):

- **Precision 0.60**: De todas las veces que predijo clase 0, acertó el 60%.
- **Recall 0.60**: De los 5 verdaderos ejemplos de clase 0, acertó 3.
- **f1-score 0.60**: Media entre precisión y recall.
- **Support 5**: Había 5 muestras reales de clase 0.

KNN Coseno

	precision	recall	F1-score	support
Taladrar	0.57	0.80	0.67	5
Martillar	0.75	0.50	0.60	6
Atornillar	1.0	1.0	1.0	6
Accuracy			0.76	17
Macro avg	0.77	0.77	0.76	17
Weighted avg	0.79	0.76	0.76	17

Tabla 4. Resultados KNN Coseno.

Clase 0

- **Precision:** 0.57 → De todas las veces que el modelo predijo clase 0, el 57% eran correctas.
- **Recall:** 0.80 → El 80% de los ejemplos reales de clase 0 fueron correctamente identificados.
- **F1-score:** 0.67 → Promedio armonizado entre precisión y recall. Es moderado, lo que sugiere que aunque se recuperaron bien los datos de esta clase, hubo algunas predicciones incorrectas.

Clase 1

- **Precision:** 0.75 → El 75% de las predicciones como clase 1 fueron correctas.
- **Recall:** 0.50 → Solo se identificó correctamente el 50% de los ejemplos reales de clase 1.
- **F1-score:** 0.60 → Algo bajo, indicando que se están perdiendo varios casos reales de clase 1.

Clase 2

- **Precision:** 1.00 → Todas las predicciones hechas como clase 2 fueron correctas.
- **Recall:** 1.00 → Todos los ejemplos reales de clase 2 fueron correctamente clasificados.
- **F1-score:** 1.00 → El mejor rendimiento posible en esta clase, perfecta precisión y recuperación.

Métricas:

- **Accuracy:** 0.76 → El modelo clasificó correctamente el 76% de los 17 ejemplos totales en el conjunto de prueba.
- **Macro avg (promedio no ponderado):**
  - **Precision/Recall/F1-score:** 0.77 / 0.77 / 0.76 → Se calcula el promedio simple entre las métricas de cada clase, útil para ver el balance entre clases.
- **Weighted avg (promedio ponderado):**
  - **Precision/Recall/F1-score:** 0.79 / 0.76 / 0.76 → Considera el número de ejemplos por clase, dando más peso a las clases más frecuentes.

LDA

	precision	recall	F1-score	support
Taladrar	0.83	1.0	0.91	5
Martillar	1.0	0.83	0.91	6
Atornillar	1.0	1.0	1.0	6
Accuracy			0.94	17
Macro avg	0.94	0.94	0.94	17
Weighted avg	0.95	0.94	0.94	17

Tabla 5. Resultados LDA.

Clase 0

- **Precision:** 0.83 → El 83% de las predicciones como clase 0 fueron correctas.
- **Recall:** 1.00 → El modelo identificó correctamente **todos** los ejemplos reales de clase 0.
- **F1-score:** 0.91 → Buen equilibrio entre precisión y recall.

Clase 1

- **Precision:** 1.00 → Todas las predicciones que fueron clasificadas como clase 1 eran correctas.
- **Recall:** 0.83 → El 83% de los verdaderos ejemplos de clase 1 fueron identificados correctamente.
- **F1-score:** 0.91 → Alto rendimiento, aunque se escaparon algunos ejemplos reales.

Clase 2

- **Precision:** 1.00 → El modelo nunca se equivocó al predecir clase 2.
- **Recall:** 1.00 → Todos los ejemplos reales de clase 2 fueron correctamente identificados.
- **F1-score:** 1.00 → Rendimiento perfecto para esta clase.

Métricas:

- **Accuracy:** 0.94 → El 94% de todas las predicciones del modelo fueron correctas.
- **Macro avg:** 0.94 en precisión, recall y F1-score → Promedio simple entre las clases, muestra un excelente rendimiento balanceado.
- **Weighted avg:** 0.95 / 0.94 / 0.94 → Promedio ponderado por el número de ejemplos de cada clase, confirma que el modelo funciona bien incluso con clases de tamaños diferentes.

Fifth-SVM

	precision	recall	F1-score	support
Taladrar	0.50	.40	0.44	5
Martillar	0.83	0.83	0.83	6
Atornillar	0.71	0.83	0.77	6
Accuracy			0.71	17
Macro avg	0.68	0.69	0.68	17
Weighted avg	0.69	0.71	0.70	17

Tabla 6. Resultados fifth-svm.

Clase 0

- **Precision: 0.50** → Solo la mitad de las predicciones hechas como clase 0 fueron correctas.
- **Recall: 0.40** → Solo el 40% de los ejemplos reales de clase 0 fueron correctamente identificados.
- **F1-score: 0.44** → Rendimiento bajo, con problemas tanto para detectar como para predecir correctamente esta clase.

Clase 1

- **Precision: 0.83** → Alto nivel de precisión, la mayoría de las predicciones como clase 1 fueron correctas.
- **Recall: 0.83** → Muy buen reconocimiento de los ejemplos reales de clase 1.
- **F1-score: 0.83** → Buen equilibrio en esta clase.

Clase 2

- **Precision: 0.71** → La mayoría de las predicciones como clase 2 fueron correctas.
- **Recall: 0.83** → Alta capacidad para identificar los ejemplos reales de clase 2.
- **F1-score: 0.77** → Rendimiento aceptable y consistente.

Métricas:

- **Accuracy (Precisión total): 0.71** → El 71% de todas las predicciones fueron correctas.
- **Macro avg:**
  - **Precision: 0.68**, **Recall: 0.69**, **F1-score: 0.68**
    → Promedio simple entre clases, indicando que el modelo tiene desempeño desigual, con clase 0 debilitando el promedio.
- **Weighted avg:**
  - **Precision: 0.69**, **Recall: 0.71**, **F1-score: 0.70**
    → Promedio ponderado según el número de muestras por clase; refleja mejor el rendimiento global al considerar el tamaño de cada clase.

Comparación de clasificadores

Clasificador	Exactitud	Precision	Recall	F1-Score
Adaboost	0.88	0.88	0.87	0.87
KNN Minkowski	0.76	0.75	0.75	0.75
KNN Coseno	0.70	0.68	0.68	0.68
LDA	0.94	0.94	0.94	0.93
Fifth-SVM	0.70	0.68	0.68	0.68

Tabla 7. Resultados clasificadores.

Matriz de confusión Adaboost

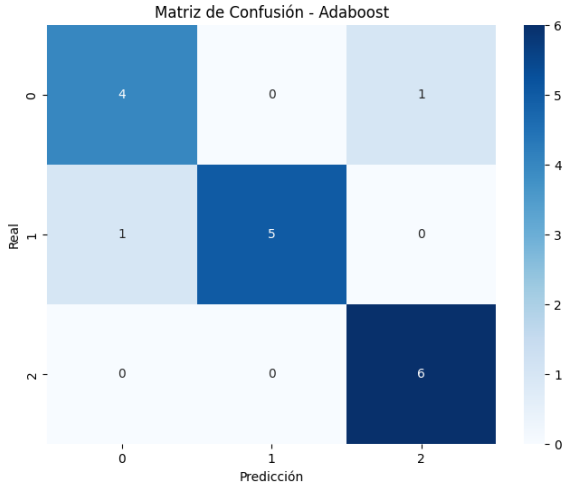

Imagen 4. Matriz de confusión adaboost.

Clase Real 0:

- **TP_00** (True Positive): El modelo predijo correctamente 4 veces la clase 0.
- **FP_01** (False Positive): El modelo predijo 0 veces que la clase era 1 cuando en realidad era 0.
- **FP_02** (False Positive): El modelo predijo 1 vez que la clase era 2 cuando en realidad era 0.

Clase Real 1:

- **FN_10** (False Negative): El modelo predijo 1 vez que la clase era 0 cuando en realidad era 1.
- **TP_11** (True Positive): El modelo predijo correctamente 5 veces la clase 1.
- **FP_12** (False Positive): El modelo no predijo ninguna vez que la clase era 2 cuando en realidad era 1.

Clase Real 2:

- **FN_20** (False Negative): El modelo no predijo ninguna vez la clase 0 cuando en realidad era 2.
- **FN_21** (False Negative): El modelo no predijo ninguna vez la clase 1 cuando en realidad era 2.
- **TP_22** (True Positive): El modelo predijo correctamente 6 veces la clase 2.

Adaboost ha mostrado un buen desempeño en este conjunto de datos, con alta precisión y recall, especialmente en la clase 2.

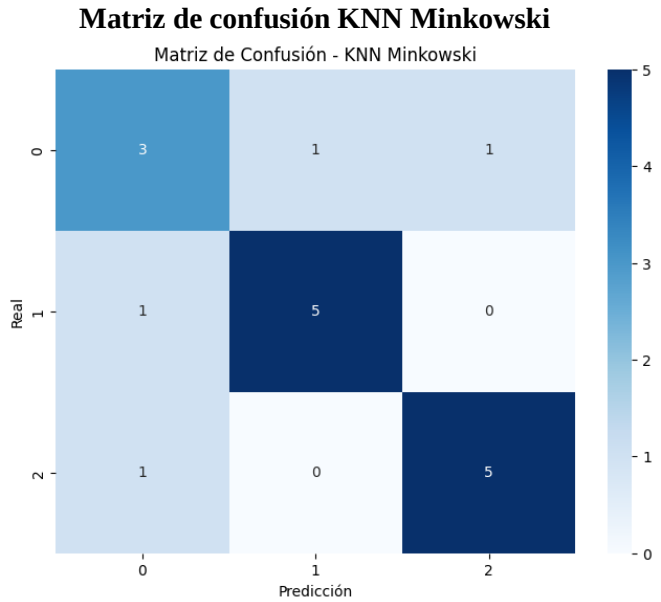

Imagen 5. Matriz de confusión KNN-Minkowski.

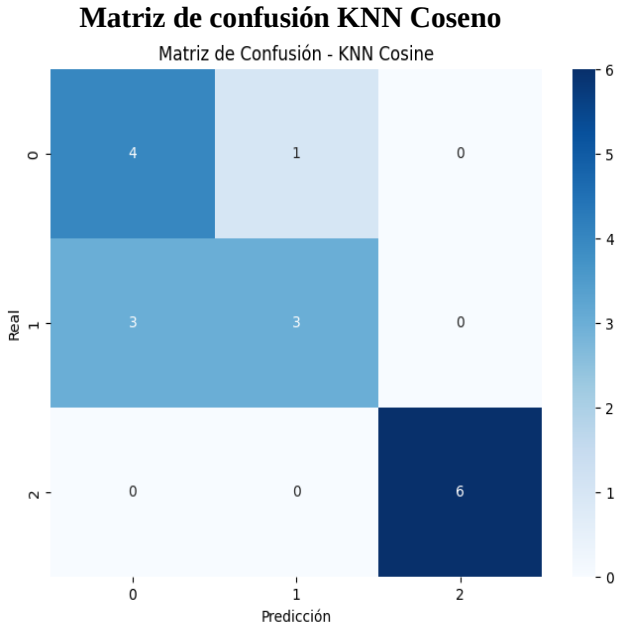

Imagen 6. Matriz de confusión KNN-coseno.

Clase 0

- **TP**: 3 veces se predijo correctamente como 0.
- **FN**: 2 errores: una vez se predijo como 1, y otra como 2.
- **Precisión** para clase 0: 3/3+1+1 = 0.60
- **Recall** para clase 0: 3/5 = 0.60

Clase 1

- **TP**: 5 veces predicho correctamente.
- **FN**: 1 caso fue confundido como clase 0.
- **FP**: No hubo falsos positivos.
- **Precisión** para clase 1: 5/5+1 = 0.83
- **Recall** para clase 1: 5/6 = 0.83

Clase 2

- **TP**: 5 veces predicho correctamente.
- **FN**: 1 caso fue clasificado erróneamente como clase 0.
- **Precisión y Recall** para clase 2: ambos 0.83

El clasificador KNN con distancia Minkowski logró un desempeño **balanceado**, con un total de **13 predicciones correctas de 17**, lo que representa una **exactitud del 76%**. Las clases 1 y 2 fueron las mejor clasificadas, mientras que la clase 0 presentó más errores de clasificación.

Clase 0

- **TP** = 4 (predichos correctamente como 0)
- **FN** = 1 (confundido como clase 1)
- **FP** = 3 (provienen de la clase 1 que fueron predichos como 0)
- **Precisión clase 0**: 4/4+3 = 0.57
- **Recall clase 0**: 4/4+1 = 0.80

Clase 1

- **TP** = 3
- **FN** = 3 (se confundieron como clase 0)
- **FP** = 1 (proviene de clase 0, predicho como 1)
- **Precisión clase 1**: 3/3+1 = 0.75
- **Recall clase 1**: 3/3+3 = 0.50

Clase 2

- **TP** = 6 (perfecto)
- **FN** = 0
- **FP** = 0
- **Precisión y Recall clase 2**: ambos = 1.00

El clasificador **KNN con métrica de Coseno** tuvo un desempeño **perfecto en la clase 2**, clasificando correctamente los 6 ejemplos. **La clase 1 fue la más conflictiva**, con 3 errores hacia clase 0 y un leve impacto en la clase 0 también.

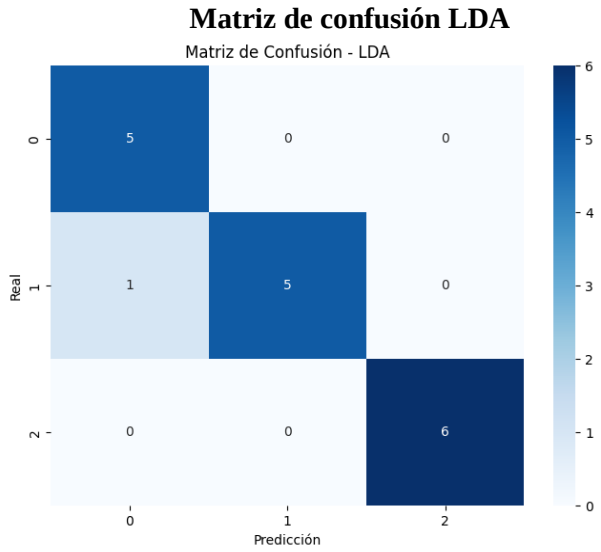

Imagen7. Matriz de confusión LDA.

Clase 0:

- **TP** = 5 (clasificados correctamente como clase 0)
- **FN** = 0 (ninguno fue mal clasificado)
- **FP** = 1 (un elemento de clase 1 fue mal clasificado como 0)
- **Precisión** = 5/5+1 = 0.833
- **Recall** = 5/5+0 = 1.0

Clase 1:

- **TP** = 5
- **FN** = 1 (un elemento fue clasificado como clase 0)
- **FP** = 0
- **Precisión** = 5/5+0 = 1.0
- **Recall** = 5/5+1 = 0.833

Clase 2:

- **TP** = 6 (todos los elementos correctamente clasificados)
- **FN** = 0
- **FP** = 0
- **Precisión** y **Recall** = 1.0

El modelo LDA tuvo un rendimiento **excelente**, con:

- Solo **un error** en toda la clasificación (una instancia de clase 1 mal clasificada como clase 0).
- Precisión y recall cercanos o iguales a 1 en todas las clases.
- Es el modelo **más equilibrado** y con mayor exactitud global entre todos los evaluados.

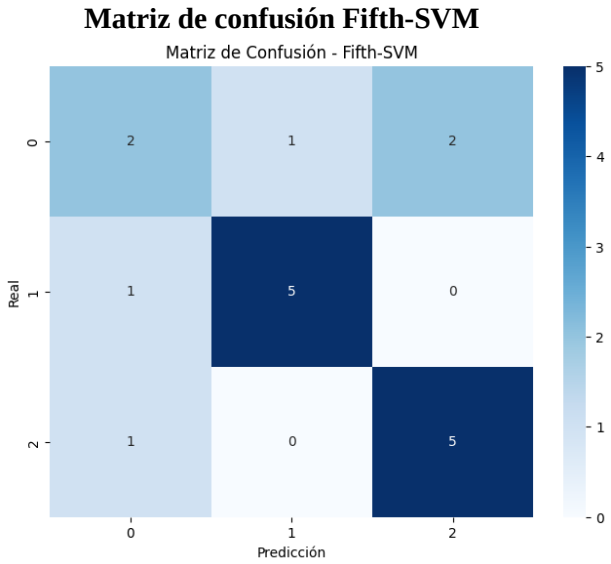

Imagen 8. Matriz de confusión fifth-svm.

Clase 0:

- **TP (Verdaderos Positivos)** = 2 (clasificados correctamente como clase 0)
- **FP (Falsos Positivos)** = 2 (uno desde clase 1, uno desde clase 2)
- **FN (Falsos Negativos)** = 3 (1 clasificado como 1 y 2 como 2)
- **Precisión** = 2/2+2=0.50
- **Recall** = 2/2+3=0.40

El modelo tuvo dificultades para identificar correctamente las instancias de la clase 0, mostrando baja precisión y recall.

Clase 1:

- **TP** = 5
- **FP** = 1 (una predicción incorrecta desde clase 0)
- **FN** = 1 (una instancia de clase 1 fue clasificada como clase 0)
- **Precisión** = 5/5+1=0.83
- **Recall** = 5/5+1=0.83

Buen desempeño en clase 1, con alta precisión y recall.

Clase 2:

- **TP** = 5
- **FP** = 2 (dos instancias mal clasificadas como clase 2 desde clase 0)
- **FN** = 1 (una instancia de clase 2 clasificada como clase 0)
- **Precisión** = 5/5+2≈0.71
- **Recall** = 5/5+1≈0.83

Desempeño aceptable, aunque con algunas confusiones con clase 0.

El modelo **Fifth-SVM (sigmoid)** tiene un rendimiento **moderado**, con una exactitud general de **71%**, el problema se encuentra en la **clase 0**, donde hubo muchas confusiones con clases 1 y 2.

IV. DISCUSION

Mejor desempeño general: LDA

- **Mayor exactitud (0.94)** y valores muy altos y balanceados en **precisión, recall y F1-score**.
- Es el modelo más efectivo para este conjunto de datos, capturando con precisión la estructura de las clases.

Segundo mejor: Adaboost

- También presenta buen desempeño general (**exactitud de 88%**), con métricas altas y balanceadas.
- Buen modelo alternativo si se desea usar un enfoque de ensamblado.

Desempeño medio: KNN Minkowski

- Resultados aceptables (exactitud ~76%), aunque inferiores a LDA y Adaboost.
- Puede mejorar con una mejor elección de **k** o una métrica de distancia más apropiada.

Menor desempeño: KNN Cosine y Fifth-SVM

- Ambos tienen resultados idénticos en las métricas globales (exactitud ~70%).
- Son los menos recomendables en esta evaluación, posiblemente por su sensibilidad a la orientación o a la forma de separación de clases.

Los resultados muestran diferencias significativas entre los modelos evaluados. Adaboost logró una precisión global del 88%, con excelente desempeño en todas las clases, especialmente en la clase 1 (100% de precisión) y la clase 2 (100% de recall), reflejando un modelo equilibrado y confiable.

KNN con métrica Minkowski alcanzó un 76% de precisión, con buen rendimiento en las clases 1 y 2 (precisión y recall de 0.83), pero con una caída en la clase 0 (60%), lo que indica sensibilidad a clases más complejas o desbalanceadas. KNN con métrica Cosine también obtuvo un 76%, destacando en la clase 2, pero con bajo recall en la clase 1 (0.50), sugiriendo que no identifica correctamente muchos ejemplos reales de esa clase.

El modelo LDA fue el más preciso, con un 94% de acierto general y excelente balance entre todas las clases. La clase 2 fue clasificada perfectamente, y las demás clases también alcanzaron altos valores de F1-score, confirmando la robustez de LDA como clasificador.

En contraste, Fifth-SVM con kernel sigmoid tuvo la menor precisión (71%), con problemas para identificar la clase 0 (recall del 40%). Esto indica que el kernel utilizado no separa bien las clases, por lo que sería conveniente probar otros kernels o ajustar sus parámetros.

V. CONCLUSION

Esta práctica me permitió comprender en profundidad el funcionamiento, ventajas y limitaciones de diversos clasificadores supervisados aplicados a un conjunto de datos con múltiples clases. A través del análisis comparativo de Adaboost, KNN (con distintas métricas), LDA y una variante SVM, se vio cómo las características del modelo, los parámetros utilizados y la naturaleza del conjunto de datos pueden influir significativamente en el rendimiento de un clasificador.

Se aprendió la importancia de evaluar el desempeño no solo con la precisión global, sino también utilizando métricas más específicas como precisión por clase, recall y F1-score, que permiten identificar debilidades puntuales del modelo, como la incapacidad para detectar ciertas clases. Asimismo, se observó cómo la elección de la métrica de distancia (en el caso de KNN) o del kernel (en SVM) puede afectar sustancialmente la capacidad de generalización del clasificador.

Esta practica reforzó la utilidad de herramientas como la matriz de confusión para visualizar de forma clara los aciertos y errores de los modelos, facilitando una interpretación más completa del rendimiento. También quedó en evidencia que modelos más simples como LDA pueden ofrecer resultados altamente competitivos si los datos se ajustan a sus supuestos.

Referencias

[1] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.