



UNCUYO
UNIVERSIDAD
NACIONAL DE CUYO



FACULTAD DE
INGENIERÍA

TRABAJO FINAL INTELIGENCIA ARTIFICIAL I 2025

**“Clasificador de piezas mediante
visión artificial controlado por
reconocimiento de voz”**

Autor: Juan Gabriel MAMANÍ MONTAÑO

Facultad de Ingeniería- UNCuyo

Cátedra: Prof. Titular Dra. Ing. Selva S. Rivera - JTP Ing. Juan I. García

RESUMEN

A continuación, se detalla el diseño e implementación de un agente cuyo objetivo es clasificar diversas piezas a través de visión artificial, además de ejecutar acciones en función de comandos de voz, así como también, estimar la distribución de las piezas dentro de las cajas de las cuales se seleccionaron las muestras.

Para poder lograr el cometido, se emplearon algoritmos de aprendizaje supervisado, como KNN para los comandos de voz, y no supervisado como lo es K-Means para el reconocimiento de imágenes. Como último paso, se aplicó una estimación de proporciones a través de aprendizaje bayesiano, para poder determinar la distribución de piezas dentro de la caja de la cual se extraen las muestras.

Fue un desafío lograr un óptimo desempeño del agente, sobre todo para el reconocimiento mediante visión. A pesar de ello se obtuvo una excelente precisión en la identificación. Tanto esta etapa, como en el desarrollo del reconocimiento de voz, se siguieron directrices de trabajo similares, aplicando conceptos tratados durante el cursado y nuevos conocimientos que se consiguieron por medio de investigación.

Los resultados obtenidos, se estiman son mejorables, aun así, la versión del agente que se presenta cumple de manera óptima con los requerimientos propuestos.

INTRODUCCIÓN

Visión Artificial

también conocida como visión por computadora, es una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica que pueda ser tratada por un ordenador. Tal y como los seres humanos usan sus ojos y cerebros para comprender el mundo que los rodea, la visión artificial trata de producir el mismo efecto para que los sistemas puedan percibir y comprender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación. Esta comprensión se consigue gracias a distintos campos como la geometría, la estadística, la física, matemática, entre otros.

La adquisición de los datos se consigue por varios medios como secuencias de imágenes, vistas desde varias cámaras de video o datos multidimensionales desde un escáner.

Hay muchas tecnologías que utilizan la visión por ordenador, entre las cuales se encuentran el reconocimiento de objetos, la detección de sucesos, la reconstrucción de una escena (mapping) y la restauración de imágenes.

El objetivo final de la visión informática es conseguir el desarrollo de estrategias automáticas para el reconocimiento de patrones complejos en imágenes de múltiples dominios

Reconocimiento de voz

El reconocimiento automático de voz es una disciplina de la inteligencia artificial que tiene como objetivo permitir la comunicación hablada entre seres humanos y computadoras. El problema que se plantea en un sistema de este tipo es permitir el intercambio de información proveniente de diversas fuentes de conocimiento (acústica, fonética, fonológica, léxica, sintáctica, semántica y pragmática), que aún en presencia de ambigüedades, incertidumbres y errores inevitables, se logra obtener una interpretación aceptable del mensaje acústico recibido.

Este sistema de reconocimiento es una herramienta computacional capaz de procesar la señal de voz emitida por el ser humano y reconocer la información contenida en esta, convirtiéndola en texto u otros formatos que permitan extraer datos de dichas señales emitidas, para su posterior procesamiento y así determinar comandos que actúan sobre un proceso. En su desarrollo intervienen diversas disciplinas, tales como: la fisiología, la acústica, la lingüística, el procesamiento de señales, la inteligencia artificial y la ciencia de la computación.

Escenario del problema a resolver

Se debe diseñar un agente inteligente con determinadas capacidades, principalmente encargado de la clasificación y decisión en un entorno de producción. Entre sus funciones se encuentran el reconocimiento de diversas piezas, en este caso de tornillos, clavos, tuercas y arandelas, por medio visión artificial. Cuenta, además, con un proceso que permite identificar comandos de voz y ejecutar diversas acciones según se solicite, entre las cuales se encuentran procesar las piezas identificadas o estimar la distribución de las mismas piezas dentro de cajas donde se encuentran almacenadas.

La situación en la que se debe desenvolver el agente parte de disponer de 4 cajas distintas. El contenido de las mismas tiene diferentes proporciones de arandelas, clavos, tornillos y tuercas. Luego, entonces, se selecciona alguna de ellas de forma aleatoria, y de la misma se extraen 10 muestras al azar.

Una vez hecho esto se procede a identificar las piezas, para ello se deben fotografiar y guardar en el almacenamiento. Posteriormente se extraen determinadas características de las piezas que ayudan a representarlas de forma única e identifiable respecto a los otros tipos de piezas. Aquí es donde se entrena al agente usando el algoritmo K-Means con las características extraídas. Con esta información el sistema debe ser capaz de reconocer nuevos elementos y clasificarlos.

Con los datos de la muestra clasificada, el agente aplica un clasificador bayesiano para estimar la distribución probable de piezas en la caja completa.

Con todas las muestras clasificadas, al sistema se le carga un archivo de audio con un comando dicho por el operario. En función al comando de voz reconocido, se determina la acción a ejecutar. Los posibles comandos son los siguientes:

- “**Proporción**” → estima la distribución de piezas para determinar la caja de origen
- “**Contar**” → identifica la cantidad de piezas que se encuentran en la muestra
- “**Salir**” → muestra los resultados obtenidos y termina la ejecución del programa

Nuevamente para reconocer el comando de voz, previamente se entrena al agente con el algoritmo KNN con distintos audios. De ellos se extraen parámetros útiles que se emplean en el entrenamiento. Con esto el agente es capaz de recibir nuevos comandos y operar con respecto a la palabra reconocida.

Para el desarrollo del trabajo y entrenamiento del agente se implementó como base de datos un conjunto de imágenes de los elementos a seleccionar y clasificar, así como también grabaciones con distintas voces. Se buscó obtener una amplia gama de variedades en las imágenes y voces para hacer más versátil al agente y lograr un óptimo desempeño.

ESPECIFICACIÓN DEL AGENTE

Tipo de agente:

El sistema se clasifica como un **agente que aprende** porque su comportamiento no está completamente definido desde el inicio, sino que mejora a partir de datos y experiencia previa. El agente es entrenado utilizando conjuntos de imágenes y audios, a partir de los cuales ajusta modelos de clasificación y estimación (por ejemplo, K-Means, KNN y estimación bayesiana).

Durante la etapa de entrenamiento, el agente procesa los datos disponibles, extrae características relevantes y construye un conocimiento interno que luego es utilizado en la fase de ejecución. Gracias a este proceso de aprendizaje, ante nuevas imágenes o audios no vistos previamente, el agente es capaz de reconocer patrones, clasificar piezas, interpretar comandos de voz y alcanzar el objetivo propuesto.

Además, el agente puede describirse mediante los cuatro componentes característicos de un agente que aprende

- Módulo de aprendizaje: ajusta los modelos internos a partir de los datos de entrenamiento, incorporando la información obtenida de imágenes y audios.
- Módulo de desempeño: utiliza los modelos entrenados para tomar decisiones, como clasificar piezas, reconocer comandos de voz o estimar la caja de origen.
- Crítico: evalúa el desempeño del agente comparando los resultados obtenidos con los esperados (por ejemplo, precisión de clasificación).
- Generador de problemas: permite incorporar nuevos datos o muestras para seguir entrenando y mejorar el rendimiento del agente.

Tabla REAS

Significado	Descripción	AGENTE
Rendimiento	Define el objetivo o criterio de éxito del agente. Es la medida que evalúa qué tan bien actúa en su entorno.	Visión: identificar correctamente las piezas muestreadas. Audio: reconocer comandos de forma inequívoca. Bayes: determinar la caja de origen más probable
Entorno	Es el medio donde el agente opera y del que recibe información a través de los sensores. Puede ser estático o dinámico, determinista o aleatorio.	Visión: set de fotografía, con iluminación controlada. Audio: sala de grabación, con el menor ruido posible.
Actuadores	Son los mecanismos o dispositivos mediante los cuales el agente interactúa con el entorno .	Versión actual: pantalla de la computadora. Ésta solo interactúa con el entorno mostrando los resultados del agente Potenciales actuadores: en el caso de implementar este agente en un entorno industrial real. En ese caso podríamos tener un brazo robótico que tenga un gripper para seleccionar las piezas. El brazo también tendrá la cámara para tomar las imágenes. El sistema dispondrá también de un sistema de grabación de comandos de voz.
Sensores	Son los elementos que perciben información del entorno . Proveen las entradas o percepciones del agente.	Visión: cámara de celular, sin filtros ni efectos. Set con iluminación uniforme Audio: micrófono de celular

Propiedades del entorno de trabajo

1. **Parcialmente observable:** El agente solo accede a una muestra reducida de 10 piezas y no percibe la caja completa. Además, la cámara y el micrófono brindan información limitada del entorno.
2. **Estocástico:** Una vez que las imágenes y audios son cargados en el sistema, el entorno queda completamente definido y no intervienen procesos aleatorios durante la ejecución del agente. Para un mismo conjunto de percepciones y un mismo comando solicitado, la respuesta del agente pertenece a un conjunto de salidas posibles previamente definidas y siempre es la misma. Por lo tanto, la salida no está sujeta al azar.
3. **Episódico:** cada ciclo de operación del agente consiste en una percepción (imagen de la muestra y señal de voz) seguida de una acción (clasificación, estimación o salida). Cada episodio es completamente independiente de los anteriores: la muestra se obtiene nuevamente al azar, la estimación bayesiana se reinicia desde cero y el comando de voz no afecta ciclos futuros. Por lo tanto, el desempeño del agente en un episodio no depende de sus acciones previas.
4. **Estático:** El entorno no cambia mientras el agente está razonando. Las imágenes y audios utilizados como percepciones permanecen constantes durante todo el procesamiento, y no existe información que varíe en tiempo real
5. **Discreto:** Las clases, comandos, cantidades y estados son finitos y toman valores puntuales, no continuos.
6. **Agente individual:** El sistema está compuesto por un solo agente que percibe, procesa y actúa de manera autónoma sin interacción con otros agentes. El agente es capaz de procesar las imágenes para poder identificarlas, extraer características de los audios para poder reconocer los comandos. Con toda esta información se puede realizar las acciones previstas, aplicando Bayes para estimar las distribuciones de las cajas.

DISEÑO DEL AGENTE: visión artificial con K-Means

Adquisición

Imágenes

Las imágenes fueron obtenidas a través de la captura de las piezas con un teléfono celular. Para poder controlar la iluminación y la distancia del lente de la cámara a la pieza, se utilizó un set de fotografía fabricado para tal propósito.

Este set fue hecho con una caja de cartón con tapa rebatible. Se perforó dicha tapa con el tamaño de la cámara, y del lado interior se colocaron tiras LED con luces de tipo fría y cálida. También se colocaron otro par de luces en las paredes. La disposición de las luces tenía el objetivo de simular una luz no puntual, que no genere sombras ni sobre-iluminación sobre la pieza. Además, se utilizó como fondo un papel de color verde lima para hacer buen contraste con las piezas metálicas

La distancia del lente de la cámara al objeto también se mantenía constante, dado por la geometría de la caja utilizada.

Al utilizar todos estos componentes lográbamos controlar el entorno completo. Esto nos permitía obtener imágenes con la mejor calidad posible, ya que se evitaba capturar “ruido” o imperfecciones en las fotos obtenidas. Con esto pudimos independizar el resultado de la captura de imágenes del correcto funcionamiento del algoritmo de pre-procesamiento y evitarnos iteraciones en el código buscando los mejores resultados, cuando el problema suele estar en la calidad de las fotografías.

Resaltamos en esta etapa la importancia que tiene la calidad de nuestra base de datos.

Preprocesamiento

Escala de grises

En nuestro caso no aplicamos una escala de grises. Lo que se hizo fue procesar la imagen para pasarlo de BGR a HSV. En nuestro caso el objetivo principal no es detectar intensidad, sino separar objeto de fondo por color.

En BGR, el color está mezclado en tres canales, en cambio, en HSV, el color o matiz (H – Hue) está separado de la saturación Saturación(S) y del Brillo (V)

Esto permite detectar el color del fondo (verde-lima) de forma mucho más robusta, y así, reducir la influencia de la iluminación, las sombras y los cambios de brillo

Filtrado y eliminación del fondo

Filtro Gaussiano: Primero aplicamos un suavizado Gaussiano: `cv2.GaussianBlur(img, (3,3), 0)`.

Esto suaviza la calidad de la imagen y reduce ruido de alta frecuencia antes de segmentar el fondo. Se evitan pequeñas variaciones de color que podrían generar errores en la segmentación del fondo.

Definición del rango de color verde: Se define `LOW_GREEN = (35, 40, 40)` y `HIGH_GREEN = (85, 255, 255)`. En el formato HSV la gama del color verde está entre H (35–85)

(aproximadamente en escala de 0-255 en lugar de 0-360°); S y V se dejan amplios para tolerar variaciones de iluminación. Luego aplicamos

```
mask_fondo = cv2.inRange(img_hsv, lower, upper)
```

Entonces, se define un rango de color verde en el espacio HSV para identificar el fondo. La función *inRange* genera una máscara binaria del fondo, la cual luego se invierte para obtener una máscara del objeto de interés en blanco, y el fondo de la imagen en negro.

Operaciones morfológicas (Open y Close): estas operaciones se aplican sobre la mascara en binario generada en el paso anterior:

MORPH_OPEN (erosión + dilatación): Se utiliza para eliminar píxeles blancos aislados y algo de ruido pequeño. Además, es útil para limpiar restos del fondo

MORPH_CLOSE (dilatación + erosión): Se Rellenan huecos dentro del objeto y pequeñas discontinuidades

En ambos casos el elemento estructurante es elíptico y pequeño (3,3), lo cual es adecuado para suavizar bordes sin deformar demasiado la geometría

Binarización

Técnicamente la imagen que hemos procesado ya es binaria, pero este paso final se realiza sobre la máscara procesada para asegurar que la imagen resultante contenga únicamente dos niveles de intensidad (0 y 255), lo que simplifica etapas posteriores de análisis y clasificación.

Reajuste de imagen

Si bien en muchos sistemas este es el primer paso que se ejecuta, en nuestro caso lo hacemos al final del preprocesamiento. Este criterio se decidió en base a pruebas realizadas y los resultados obtenidos.

Por lo tanto, se ajusta el tamaño a 512x512 al final del proceso para preservar al máximo la información original durante la segmentación. Al tratarse de una imagen binaria, se utiliza interpolación por vecino más cercano para evitar la generación de valores intermedios(grises).

Segmentación

Una vez obtenidas las imágenes en binario se almacenaron en la carpeta “data/raso/binario”. Es importante destacar que para poder segmentar las imágenes y poder extraer los contornos principales, las imágenes en binario deben estar en formato png, ya que otros tipos de formato conducen a una segmentación poco precisa.

Entonces en esta etapa partimos de imágenes binarias. Con OpenCV detectamos contornos usando *findContours* con *RETR_TREE* para conservar jerarquía y así poder medir agujeros internos. Esto es clave para piezas como arandelas/tuercas, que tienen agujero interno. Si no usáramos jerarquía, aún podríamos detectar el borde externo de las piezas, pero perderíamos la relación con el contorno interno.

Luego de determinados los bordes, filtramos ruido descartando contornos muy pequeños y nos quedamos solo con contornos externos como objeto principal.

Extracción

Las características que se extraen de los contornos se definen en el archivo “features.py”. Mas específicamente con la función “extraer_features_objeto ()” que construye un diccionario con las características.

En esta etapa cada objeto segmentado se transforma en un vector de características que describen tamaño, forma, complejidad y estructura interna. Estas variables permiten que el modelo aprenda patrones discriminantes entre clases sin depender de la orientación o el tamaño de la imagen.

A continuación, se describen las características extraídas:

1-Área y área normalizada: representan respectivamente la cantidad de píxeles que ocupa el contorno del objeto; y el área del objeto dividida por el área total de la imagen.

Se normaliza para que el valor no dependa de la resolución ni del tamaño de la imagen, permitiendo comparar objetos fotografiados a distintas escalas. Permite distinguir objetos macizos de objetos delgados.

2-Perímetro y perímetro normalizado: representan longitud total del borde del objeto; y el mismo valor dividido por una escala proporcional al tamaño de la imagen.

Un objeto con muchos detalles o irregularidades tendrá mayor perímetro. Esto ayuda a diferenciar clavos (bordes simples) tornillos (bordes más complejos por la rosca)

3-Aspect Ratio: Relación entre el lado mayor y el menor del rectángulo mínimo que encierra al objeto. Entonces un valor cercano a 1 indica que el objeto es aproximadamente cuadrado o circular(tuercas/arandelas), mientras que valores altos indican objetos alargados(clavos/tornillos).

4-Extent: Relación entre el área real del objeto y el área de su “bounding box” (rectángulo envolvente alineado a los ejes). Su valor indica si el objeto ocupa bastante área de su caja, como son tuercas/arandelas, o si por el contrario es un objeto mas esbelto con un valor bajo de Extent, como ocurre con clavos/tornillos

5-Solidity: Relación entre el área del objeto y el área de su envolvente convexa. Útil para detectar piezas con agujeros de piezas sólidas. Como las tuercas y arandelas tienen un agujero, este valor siempre es menor que 1, en cambio, clavos y tornillos son mas macizos y este parámetro tiene valor cercano a 1.

6-Circularidad: Mide qué tan similar es el contorno a un círculo perfecto. Entonces piezas como las arandelas tendrán un valor cercano a 1, mientras que piezas esbeltas tendrán un valor muy bajo.

7-Número de agujeros: cantidad de contornos internos, los que determinamos con la herencia al buscar los contornos (hijos en la jerarquía). Es una característica estructural clave, muy determinante para diferenciar ciertas clases.

8-Proporción de área de agujeros: área total de los agujeros dividida por el área del objeto. Aporta información más relevante que solo contar agujeros. Permite diferenciar entre arandelas (agujero generalmente más grande) de tuercas

9-Número de vértices: Cantidad de vértices tras aproximar el contorno con el algoritmo de Ramer–Douglas–Peucker. Una pieza con pocos vértices indica una forma más simple, en cambio, muchos vértices indican una forma compleja. Dentro del modelo es útil para interpretar si la pieza se trata de un tornillo (muchos vértices por la rosca), clavos (menos vértices), o arandelas (pocos vértices por su forma circular).

10-Relación vértices/perímetro: Normaliza la complejidad del contorno respecto a su tamaño. Con esto se evita que el tamaño del objeto domine la clasificación y permite comparar complejidad relativa entre objetos grandes y chicos.

11-Momentos invariantes de Hu (Hu1 a Hu6): Los momentos de Hu son descriptores clásicos de forma, que tienden a ser invariantes ante rotación, traslación y escala; por eso ayudan, aunque la pieza esté girada. En este caso sus valores se normalizan en escala logarítmica para evitar valores extremos. Se calculan los 6 momentos para determinar a posterior cuales son los más significativos al momento de entrenar el agente.

Una vez completadas las etapas mencionadas, la información obtenida para cada objeto detectado se organiza en un dataset estructurado en formato CSV, utilizando el programa *features_to_csv.py*. En este punto, cada objeto identificado dentro de una imagen binaria es representado mediante un diccionario de características (features), donde cada clave corresponde a una magnitud calculada, y cada valor es el resultado numérico asociado a dicho objeto.

El proceso consiste en procesar todas las imágenes binarias almacenadas en el directorio de entrada correspondiente. Para cada imagen, se ejecuta la función de extracción de features, la cual puede devolver uno o varios diccionarios, dependiendo de la cantidad de objetos válidos segmentados en esa imagen. Cada diccionario se interpreta como una observación independiente del dataset, y se transforma directamente en una fila del archivo CSV. Esta conversión de diccionarios a filas permite mantener el orden entre las características calculadas y su representación en formato tabla, lo que facilita su posterior análisis o utilización en algoritmos de aprendizaje.

Se incorporan además dos columnas fundamentales: `object_id` y `clase_real`. El campo `object_id` identifica de manera única a cada objeto dentro de una misma imagen, lo cual resulta especialmente importante cuando una imagen contiene más de un objeto segmentado. Este identificador permite analizar individualmente cada pieza detectada, asegurando trazabilidad durante las etapas de validación y entrenamiento de modelos. Por otro lado, el campo `clase_real` representa la etiqueta verdadera del objeto, inferida a partir del nombre del archivo. Esta etiqueta solo se usa para etiquetar los clústeres que se forman durante el entrenamiento en base a la mayor cantidad de etiquetas en cada nube de puntos. Es decir, no se utiliza para entrenar al agente en sí, ya que el algoritmo k-means es NO supervisado, sino solo para nombrar cada clúster.

Finalmente, el archivo CSV generado contiene toda la información relevante de cada pieza, que incluye el conjunto completo de características geométricas y morfológicas. Este formato tabular resulta adecuado para su uso en herramientas de análisis de datos y permite su tratamiento directo con librerías como Pandas, o de visualización y análisis estadístico.

De esta manera, el CSV actúa como el nexo final entre el procesamiento de imágenes y la etapa de modelado, garantizando reproducibilidad, claridad y escalabilidad del proyecto.

Clasificación

Entrenamiento de K-MEANS

Implementado en el archivo `train_kmeans_manual.py`. Este es el script de entrenamiento que toma un `features.csv`, aprende los centroides con KMeans, construye un mapeo tipo clúster (nube de puntos) → clase (etiquetada), evalúa y guarda todo en disco.

Este script asume que existe un CSV con columnas numéricas (`features`) y una columna de referencia llamada `clase_real`. Si falta, corta con error. Además, filtra filas cuya `clase_real` sea "desconocida". Si hay muestras mal rotuladas o "ruido", entrenar/interpretar con esas filas confunde el mapeo clúster → clase. Sacarlas mejora consistencia.

Como KMeans es no supervisado, agrupa sin mirar etiquetas reales. Entonces el entrenamiento aprende clústeres puramente por geometría/forma. Luego usa la columna `clase_real` solo para interpretar qué significa cada clúster (por mayoría), sin cambiar el entrenamiento del KMeans.

Para la selección de `features` se propuso un conjunto razonable de descriptores morfológicos, como lo son aspect ratio, extent, Solidity, circularity, agujeros, momentos de Hu (Hu1, Hu2, Hu3). Como este algoritmo trabaja en un espacio vectorial, mientras más informativas las dimensiones, mejor se separan clases.

Antes de proceder con el entrenamiento en sí se hace un escalado de los valores numéricos de las propiedades de las piezas. Para ello se utiliza un `scaler`, ya que este algoritmo usa distancias euclídeas. Entonces si una característica tiene rango 0–10000 y otra 0–1, la grande domina y el clustering puede ignorar lo demás. Entonces este `scaler` aplica:

- media μ por columna
- desvío σ por columna
- transforma: $z = (x - \mu) / \sigma$

Para crear el modelo entrenado se elige `n_clusters=4`, porque este problema tiene 4 clases (arandela, clavo, tornillo, tuerca). Los centros iniciales se eligen de puntos aleatorios del dataset. Se repite el algoritmo 20 veces con semillas distintas y conservamos aquella con la mejor solución (menor inercia).

Se establece una máxima cantidad de iteraciones igual a 300 y/o una tolerancia de convergencia igual a 0,0001

Luego de entrenar se obtienen los clústeres etiquetados como (0,1,2,3). Para poder interpretar a que pieza corresponde cada uno, para cada clúster c se observa qué clase real aparece más dentro del clúster. Entonces Asignamos ese clúster a esa pieza. En este paso debemos distinguir que no estamos entrenando un clasificador supervisado. Estamos nombrando clústeres con una regla posterior para poder evaluar y usar en la predicción de nuevas piezas.

El entrenamiento del modelo no es solo almacena los centroides, sino que también contiene información sobre qué características(features) se usaron para entrenar, cómo se escalan, dónde están los centroides, y cómo se nombran a cada clúster según la pieza.

Implementación matemática del KMeans

KMeans busca centroides $C = \{c_1, c_2, c_3, c_4\}$ que minimicen la suma de distancias cuadradas dentro del clúster, nosotros lo calculamos como:

$$\text{inercia} = \sum_{i=1}^N \|x_i - c_{label(i)}\|^2$$

Luego para las distancias cuadradas armamos una matriz (N, K) con todas las distancias $\|x_i - C_j\|^2$, esto es equivalente a computar la distancia euclídea, solo que sin raíz (la raíz no cambia el argumento de la distancia)

Para inicializar los centroides se elige K muestras (aleatorias) distintas del dataset como centros iniciales. Durante un ciclo de iteración del algoritmo (función `run_once ()`) se calculan las distancias, entonces cada punto va a centro mas cercano. Luego para cada clúster k , se recalcula el nuevo centro como la media de los puntos asignados (aplicamos el algoritmo de Lloyd).

Si un clúster no recibe puntos, resembramos con un punto aleatorio del dataset, ya que de no hacerlo así ese centro queda inválido y rompe el algoritmo. Por lo tanto, esta nueva semilla mantiene los clústeres activos y permite que el algoritmo siga.

Con respecto a los criterios de parada, para detectar convergencia, usamos dos “frenos”. Uno es simplemente evaluar que la diferencia entre la distancia actual al centroide y la distancia previa sea menor que la tolerancia establecida. El otro criterio esta relacionado con la posición del centroide, entonces si en dos iteraciones sucesivas el cambio en su posición es menor que la tolerancia, el proceso también se detiene.

Para la elección del mejor resultado, `fit ()` corre `_run_once` varias veces (hasta `n_init`) con semillas distintas y se queda con la menor inercia.

Una vez que esta todo ajustado, para clasificar nuevas imágenes la función `predict ()` solo calcula las distancias a centros finales y devuelve el índice del centro más cercano.

Lo último que se realiza durante el entrenamiento es evaluar que tan bien clasifica la imágenes con la cual se entrenó el agente. Se calcula una matriz de predicción vs real. De esta matriz se obtiene que la precisión alcanzada es alrededor del 94%.

Interpretación

Una vez entrenado el agente y almacenados en memoria todos los modelos, la predicción de nuevas imágenes se implementa en el archivo `predecir_k4_manual ()`. Este archivo es el que se

utiliza para clasificar una imagen nueva con el KMeans entrenado y guardado. Lo importante es que reutiliza exactamente el mismo flujo de extracción de características.

El proceso empieza por leer la imagen (muestra a identificar) con OpenCV. Si la imagen ya es 2D (o tiene un canal), asume que es una máscara binaria. Si no, llama a *binarizar ()*. Esta sección del agente no trabaja sobre pixeles sin procesar de la imagen, sino que trabaja sobre descriptores de las piezas, por eso primero se necesita binarizar/segmentar el objeto. Entonces se extraen las características del objeto con las funciones ya mencionadas en la etapa de preprocesamiento.

Luego se cargan los modelos entrenados para poder usarlos en la clasificación. Como el sistema se entrenó sobre datos escalados, hay que escalar igual manera en inferencia de la pieza tratada. Además, como solo se identifica el clúster más cercano, se necesita mapping para convertir a clase con etiqueta de la pieza.

Un aspecto que es importante es el orden de los features, debemos cargar los mismos en el orden con el cual se entrenó. Si en la inferencia se permutan columnas, la distancia euclídea cambia y el clúster asignado será incorrecto.

Por último, la función *predict ()* devuelve un número de clúster (0, 1, 2 o 3). Luego ese clúster se traduce a clase con el mapping aprendido por la mayor cantidad de etiquetas reales en cada nube de puntos (*majority vote*).

DISEÑO DEL AGENTE: reconocimiento de voz con KNN

Adquisición

Audios

Aplicando todo lo aprendido en la construcción de la base de datos de imágenes, lo primero que se hizo en este caso fue investigar sobre las condiciones óptimas en las que se debían grabar los audios.

Luego de consultar distintas fuentes, se determinó que los audios serían capturados con el micrófono del teléfono y con ciertas especificaciones que aseguran que la información contenida en la señal sea fiel al comando registrado y sea adecuada para su posterior análisis.

Se consideró capturar los audios en formato WAV (Waveform Audio File Format), utilizando codificación PCM (Pulse Code Modulation) sin compresión, ya que esto nos permite mantener íntegramente la señal original. Además, de esta forma se facilita la extracción de características temporales y espectrales.

La frecuencia de muestreo seleccionada para la grabación de los audios fue de 16.000 Hz (16 kHz). Esta elección responde a un criterio estándar en aplicaciones de reconocimiento de voz. Según el teorema de Nyquist, una frecuencia de muestreo de 16 kHz permite representar correctamente señales con contenido espectral de hasta 8 kHz, rango suficiente para capturar la información relevante de la voz humana. Esto reduce el tamaño de los archivos de audio y el costo computacional en comparación con frecuencias más altas, sin comprometer la información necesaria para la tarea.

Los audios se grabaron en modo mono, es decir, utilizando un único canal de audio. Debido que la información relevante para el reconocimiento se encuentra en el contenido acústico de la señal y no en su espacialidad, el uso de un solo canal simplifica el procesamiento y la representación de los datos. También se reduce el tamaño de los archivos y evita inconsistencias entre canales izquierdo y derecho.

Cada muestra de audio presenta una duración aproximada de entre 1 y 2 segundos, manteniéndose una longitud similar entre los distintos archivos. Las grabaciones se realizaron bajo condiciones controladas, procurando minimizar la influencia de factores externos que pudieran introducir ruido o variabilidad innecesaria en los datos.

Preprocesamiento

El preprocesamiento de la señal de voz se implementó en el archivo preprocess_audio.py, mediante la función preprocess_audio_file (). Esta función recibe un audio original(input_path) y genera un audio limpio (output_path) listo para extraer características.

El proceso empieza cuando se carga el audio en un solo canal(mono) y en caso de que la grabación no tenga esta característica (por lo general los audios de prueba), se fuerza que la frecuencia de muestreo sea de 16000 Hz, para que todas las muestras tengan el mismo dominio temporal, evitando inconsistencias entre audios

Para reducir ruido y evitar que el modelo tenga en cuenta posibles silencios, se recorta el audio usando librosa.effects.trim con un umbral top_db=20. Con esto se elimina gran parte del silencio al inicio y final, quedando principalmente la palabra pronunciada

Para normalizar la amplitud se divide la señal por su valor absoluto máximo (teniendo en cuenta una epsilon pequeña para evitar división por cero) para que todas las grabaciones queden comparables en escala de amplitud. Esto es importante porque distintas voces generan niveles distintos de amplitud. Luego se aplica un filtro de preénfasis con coeficiente 0.97. Este paso realza componentes de alta frecuencia, ayudando a que ciertos rasgos de la voz sean más distinguibles en la etapa de extracción de características.

Finalmente, el audio procesado de guarda también en formato WAV. Este mismo proceso de aplica a todos los audios crudos y luego de aplicar los pasos mencionados se guardan en otro directorio para poder extraer datos de esas señales.

Segmentación

En este proyecto, cada archivo de entrenamiento contiene un único comando (“proporción”, “contar” o “salir”), de alrededor de 1–2 segundos. Por lo tanto, no se implementó una segmentación por ventanas que constaría de dividir en frames y detectar dónde comienza/finaliza el habla, sino más bien, tendremos una segmentación práctica basada en recorte de silencios (trim) como mecanismo de detección del fragmento relevante de voz, manteniendo la señal resultante completa como una sola unidad para describirla mediante estadísticas (promedios y desvíos) en la extracción de características.

Por lo tanto, en lugar de cortar explícitamente la palabra por timestamps, se reduce el audio al tramo donde hay energía de voz, y sobre ese tramo se calculan las características.

Extracción

La extracción de características se implementó en `features_audio.py` mediante la función `extract_features_from_file()`. El enfoque a seguir aquí es convertir la señal (vector temporal) en un vector numérico fijo con las mismas columnas para todos los audios, que capture información relevante de la voz.

Por tanto, primero se cargan los audios ya preprocesados en la etapa anterior, en canal mono y la misma tasa de muestreo. Si el audio está vacío se descarta para no producir errores en el proceso.

A continuación, se describen las características extraídas de las señales de audio:

MFCC (Mel-Frequency Cepstral Coefficients): son coeficientes que describen la envolvente espectral del audio en una escala perceptual (Mel), que imita cómo el oído humano percibe las frecuencias. Para obtener estos coeficientes se sigue un proceso específico, donde se aplica la transformada rápida de Fourier para obtener la señal en el dominio de la frecuencia, se aplica un filtro en escala Mel, luego se pasa a escala logarítmica (como percibe el oído humano), se aplica Transformada discreta del coseno (DCT), finalmente se obtienen coeficientes MFCC (1 a 13), que posteriormente normalizamos con la media y desvío estándar.

Estos coeficientes dan información acerca de los fonemas, vocales, timbre del hablante y son robustos frente a ruido moderado

ZCR (Zero Crossing Rate): la tasa de cruces por cero (cambios de signo) ayuda a distinguir señales más ruidosas o con distinta estructura temporal. Según el tipo de señal un valor de ZCR alto indica sonidos fricativos, ruido, consonantes (/s/, /f/), por otro lado, un ZCR bajo representa sonidos vocales, graves, sonoros.

RMS (energía): Mide la energía efectiva de la señal. No es simplemente amplitud máxima, sino, el nivel de potencia promedio del audio. Es un parámetro importante debido a que permite detectar silencios, diferenciar comandos cortos o largos, y separar palabras con distinta energía.

Spectral Centroid: mide el “centro de masa” del espectro, es decir, qué tan cargado a frecuencias altas/bajas está el sonido. Es útil para diferenciar entre vocales graves (/a/, /o/), consonantes agudas (/s/, /sh/), y cambios de timbre entre palabras

Spectral Bandwidth: mide dispersión del espectro alrededor del centroide. Si el sonido está concentrado en pocas frecuencias (voz sostenida) o distribuido en muchas frecuencias (ruido, fricción)

Una vez extraídas las características de cada audio se construye nuestro dataset de entrenamiento con la función `build_dataset()`. Ésta recorre una carpeta de audios WAV, extrae features y construye un CSV con columnas `file_path` (trazabilidad), `label` (clase real del comando), y todas las features extraídas

La etiqueta no se escribe a mano, sino que se infiere del nombre del archivo, tomando letras desde el inicio hasta antes del primer dígito. Se implementa con una regex (`re.match`) que admite letras incluyendo acentos y ñ. Ejemplos: `proporcion01.wav` → `proporcion`, `salir15_prueba.wav` → `salir`. Finalmente se guarda el CSV en `data_voz/features/features_audio.csv`

Clasificación

El entrenamiento se implementó en `src/_knn_manual.py` en la función `train_knn()`

La lógica general es comienza con la preparación de datos (`X, y`), para lo cual se lee el CSV de `features`, y se separa en dos partes:

`X`: todas las columnas numéricas (features), eliminando `file_path` y `label`

`y`: vector de etiquetas (`label`)

Además, se guarda `feature_names = list(X.columns)` para conservar el orden exacto de columnas usado en el entrenamiento. Esto es importante luego al momento de inferir el comando en nuevos audios.

Para el entrenamiento en sí, se dividió el dataset en archivos de entrenamiento y de prueba, en una proporción 70-30, respectivamente, del total de audios en la base de datos. Esto es una mejora respecto a lo implementado en K-Means, ya que en aquel algoritmo se evaluó la precisión con una especie de doble evaluación sobre los datos de entrenamiento.

Como KNN depende completamente de distancias, las features deben estar en escalas comparables. Por eso se aplica un escalador sobre las características extraídas. Esto evita que, por ejemplo, una feature con rango grande domine la distancia euclídea.

Para el modelo se determinaron los `k` vecinos más cercanos con un valor de `k=5`. La métrica de distancia será euclídea, y la ponderación por distancia se hará con los pesos de cada una de ellas, es decir, La contribución de cada vecino se pondrá según la inversa de su distancia, otorgando mayor influencia a los vecinos más cercanos.

En este caso determinamos KNN no entrena ajustando parámetros como en KMeans; su etapa `fit` solo almacena los ejemplos etiquetados. El costo real aparece en `predict`, porque ahí se calculan distancias respecto a todo el conjunto de características. Entonces lo que hacemos con `fit` es que el vector `X` con las características quede normalizado y consistente con dimensión `n`, para que después puede ser usado en la inferencia. Entonces este modelo aprende observando ejemplos ya etiquetados. Lo que se usa como referencia son

los ejemplos reales. Entonces cuando llega un audio nuevo, se compara contra todas (o muchas) las muestras guardadas.

Por lo tanto, nosotros tendremos en una matriz X con vectores que contienen los valores de los features, por un lado, y por el otro, una matriz y que tiene vectores con las etiquetas de esos.

Entonces la etapa fit (X, y) cumple las siguientes funciones:

Validar la consistencia de los datos

- X debe ser una matriz de dimensión (N, m), donde:
 - o N es la cantidad de muestras (audios).
 - o m es la cantidad de características extraídas por audio.
- y debe ser un vector de longitud N, que contenga la etiqueta correspondiente a cada muestra.

Almacenar los datos de entrenamiento: se guardan internamente:

- X_{train} : matriz de vectores de características.
- y_{train} : vector de etiquetas asociadas.

Preparar el conjunto de referencia: no se calculan centroides ni parámetros. El conjunto de entrenamiento completo actúa como referencia para la etapa de inferencia.

Por lo tanto, entrenar un KNN equivale a aprender ejemplos correctamente representados, por eso es un algoritmo supervisado.

Entonces cuando llega un audio nuevo, del él se extraen sus features y se arma un vector x. Luego con cada uno de los valores de este vector se calcula la distancia euclídea con la siguiente fórmula:

$$d(x, x_i) = \sqrt{\sum_{j=1}^n (x_j - x_{i,j})^2}$$

Notar que se calcula esta distancia respecto a cada uno de los audios del entrenamiento. Por lo tanto, se obtiene un valor de distancia por cada audio. Aquí es donde notamos la importancia de haber normalizado los valores y, sobre todo, de tener los features bien ordenados, ya que, si no fuera así, se estaría calculando una distancia entre features que no son homólogos, lo cual no tendría sentido y rompe la lógica.

Luego, estas distancias calculadas se ordenan en forma ascendente, y se toman los primeros 5 valores de di, es decir, los más cercanos. Dado que no todos los vecinos aportan la misma información, se utiliza una ponderación basada en la inversa de la distancia:

$$w_i = \frac{1}{d(x, x_i) + \epsilon}$$

- Si la distancia es pequeña → peso grande
- Si la distancia es grande → peso chico
- ϵ evita división por cero cuando hay coincidencia exacta.

Este esquema garantiza que los vecinos más cercanos tengan mayor influencia; y que los vecinos más lejanos contribuyan en menor medida.

Y la clase final se decide por suma de pesos por clase:

$$\hat{y} = \arg \max_c \sum_{i \in N_k(x)} w_i \cdot \mathbf{1}(y_i = c)$$

Donde $\mathbf{1}(y_i=c)$ es una función indicadora que vale:

- 1 si la etiqueta del vecino i es la clase c ; $c=[\text{"contar"}, \text{"proporcion"}, \text{"salir"}]$
- 0 en caso contrario.

Es decir, se suman todos los pesos de las etiquetas que correspondan a “contar”, luego las que correspondan a “proporcion”, y luego las de “salir”. La etiqueta que sume el mayor peso, se designa como etiqueta final para el comando reconocido

Interpretación

La etapa de interpretación corresponde al proceso mediante el cual el agente transforma una señal de audio capturada en una decisión concreta que modifica su comportamiento. Esta fase se implementa en el archivo predict_knn_voice_manual.py, a través de la función predict_command, y constituye la fase de inferencia del sistema de reconocimiento de voz basado en KNN.

Antes de realizar cualquier inferencia, el agente verifica la existencia del modelo KNN previamente entrenado, el escalador utilizado durante el entrenamiento y la lista de nombres de características. Esta verificación es fundamental para garantizar la coherencia entre la etapa de entrenamiento y la de predicción, ya que la ausencia de alguno de estos elementos impediría realizar una clasificación confiable.

Luego el nuevo audio de entrada, es sometido al mismo proceso de preprocessamiento aplicado durante la construcción del conjunto de entrenamiento. Para ello, el audio es normalizado, recortado para eliminar silencios y filtrado mediante preénfasis, generando un archivo temporal que asegura condiciones equivalentes a las de los datos utilizados para entrenar el modelo.

Sobre el audio preprocessado se ejecuta la extracción de características acústicas. El sistema obtiene un conjunto de descriptores numéricos. El resultado de esta etapa es un diccionario de características que describe de manera compacta la señal de voz del comando.

Con el vector de features se realizan todos los pasos descriptos en la sección del modelo del KNN. Finalmente, la etiqueta predicha es interpretada como una orden semántica dentro del agente. Cada clase reconocida corresponde a un comando específico previamente definido en el diseño del sistema, como iniciar el cálculo de proporciones, realizar el conteo de piezas o finalizar la ejecución del agente. De este modo, el reconocimiento de voz no se limita a identificar una palabra, sino que se integra directamente al ciclo de decisión del agente

DISEÑO DEL AGENTE: aprendizaje bayesiano

La estimación bayesiana se utiliza como etapa final de decisión del agente, con el objetivo de inferir la caja de origen más probable a partir de una muestra finita de piezas previamente clasificadas por el módulo de visión artificial. Dado que el agente solo observa una fracción del contenido total, el problema se aborda como un proceso de inferencia probabilística bajo incertidumbre.

Formulación del problema

Se dispone de un conjunto finito de hipótesis $H = \{C_A, C_B, C_C, C_D\}$, donde cada hipótesis representa una caja posible. Para cada caja se conoce la distribución condicional de piezas:

$$P(p | Ci); \quad p \in \{\text{tornillo, clavo, arandela, tuerca}\}$$

Estas probabilidades constituyen el conocimiento previo del sistema y modelan la composición interna de cada caja. Inicialmente se asume que a priori, antes de observar piezas, todas las cajas son igualmente probables, ya que no se dispone de otra información que indique una probabilidad inicial para cada caja.

Actualización bayesiana secuencial

Sea p_k la pieza observada en la extracción de muestra número k. El agente actualiza su creencia aplicando el teorema de Bayes:

$$P(C_i | p_k) = \frac{P(p_k | C_i) P(C_i)}{\sum_j P(p_k | C_j) P(C_j)}$$

En la implementación, el numerador se calcula para cada caja y luego se normaliza la distribución para asegurar que:

$$\sum_i P(C_i | p_k) = 1$$

Esta verificación se hace para asegurar que el cálculo y actualización de probabilidad sea correcto en cada extracción de una piza.

La distribución posterior obtenida tras observar una pieza se utiliza como prior para la siguiente observación. De esta forma, para una muestra ordenada $\{p_1, p_2, \dots, p_N\}$, el agente realiza una actualización iterativa:

$$P(C_i | p_1, \dots, p_N) \propto P(C_i) \prod_{k=1}^N P(p_k | C_i)$$

Este enfoque permite modelar el razonamiento incremental del agente, donde cada nueva pieza refina la estimación previa. Sería como aplicar el teorema de Bayes de forma interactiva.

Entradas y salidas del módulo

La entrada del módulo bayesiano es una lista de etiquetas de piezas clasificadas por K-Means. La salida es una distribución de probabilidad sobre las cajas, no una decisión determinística. El agente selecciona como hipótesis principal aquella caja con mayor probabilidad posterior, pero conserva la distribución completa como medida explícita de incertidumbre.:

$$C^* = \arg \max_{C_i} P(C_i | \text{muestra})$$

A diferencia de K-Means y KNN, que operan como clasificadores directos, el módulo bayesiano actúa como un integrador de información y un decisor probabilístico. Permite combinar conocimiento previo del entorno con evidencia observada, cerrando el ciclo percepción–decisión del agente de manera formal y coherente desde el punto de vista estadístico.

CÓDIGO

El proyecto fue desarrollado siguiendo una organización modular, separando claramente los datos de entrada, los algoritmos de aprendizaje y el agente principal que integra todo el sistema. El código completo se encuentra publicado en un repositorio público, lo que permite su análisis y ejecución por parte de los docentes.

El repositorio del proyecto puede consultarse en:

https://github.com/Gabrielle-23/clasificador_piezas_ia_2025

Estructura general del proyecto

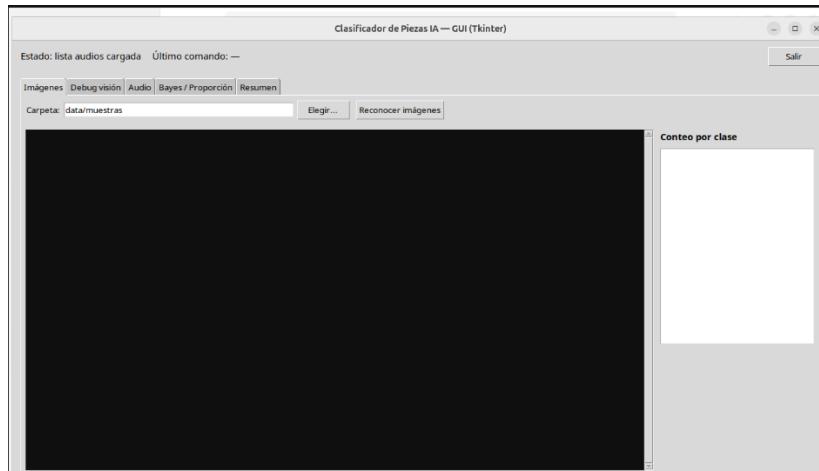
A continuación, se presenta el árbol de directorios que resume la organización del proyecto:

```
|- agente.py
|- arbol_directorios.txt
|- data
|   |-- features
|   |-- models_k4_manual
|   |-- muestras
|   |-- pruebas
|   |-- raso
|- data_voz
|   |-- features
|   |-- interm
|   |-- models_manual
|   |-- muestras_audio
|   |-- pruebas
|   |-- raso
|   `-- tmp_preprocessed.wav
|- main.py
|- procesamiento.ipynb
|- README.md
|- requirements.txt
|- src
|   |-- bayes.py
|   |-- binario.py
|   |-- contornos.py
|   |-- features_audio.py
|   |-- features.py
|   |-- features_to_csv.py
|   |-- filtro.py
|   |-- kmeans_manual.py
|   |-- knn_manual.py
|   |-- ml_manual
|   |-- predecir_audio_manual.py
|   |-- predecir_k4_manual.py
|   |-- predecir.py
|   |-- preprocess_audio.py
|   |-- train_kmeans_k4_manual.py
|   |-- utils.py
|- Tabla Reas.xlsx
|- TRABAJO FINAL.docx
```

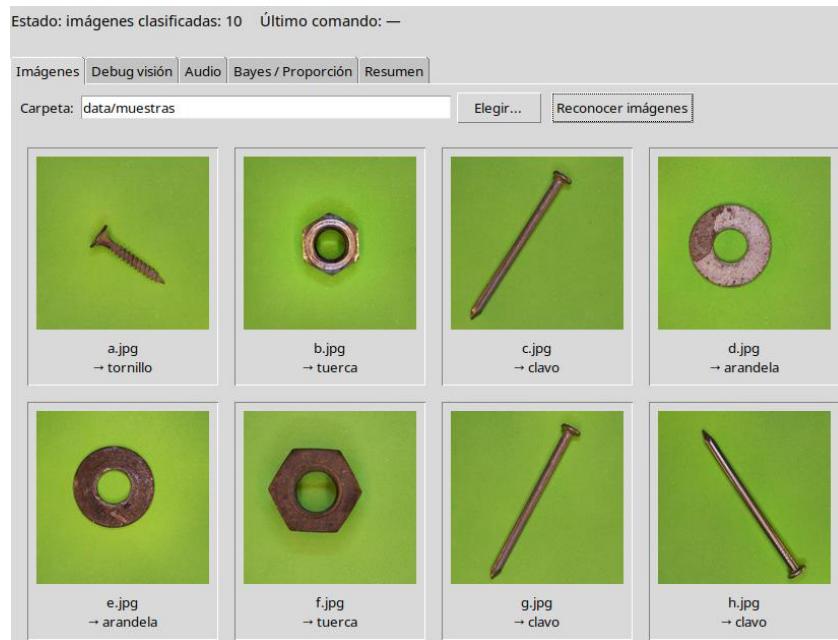
Las carpetas “data” y “data_voz” contienen las bases de datos con los archivos crudos utilizados para el entrenamiento y validación del sistema. El procesamiento y la lógica de clasificación se implementan íntegramente dentro de la carpeta src/.

EJEMPLO DE APLICACIÓN

Tenemos dos programas que se pueden ejecutar para mostrar el funcionamiento del agente. En este caso mostraremos el flujo de trabajo del archivo “gui.py”. Al ejecutar este archivo en la terminal de Ubuntu/Linux, se muestra la siguiente ventana. Aquí podemos ver que tenemos pestañas para ver cada etapa de trabajo del agente.

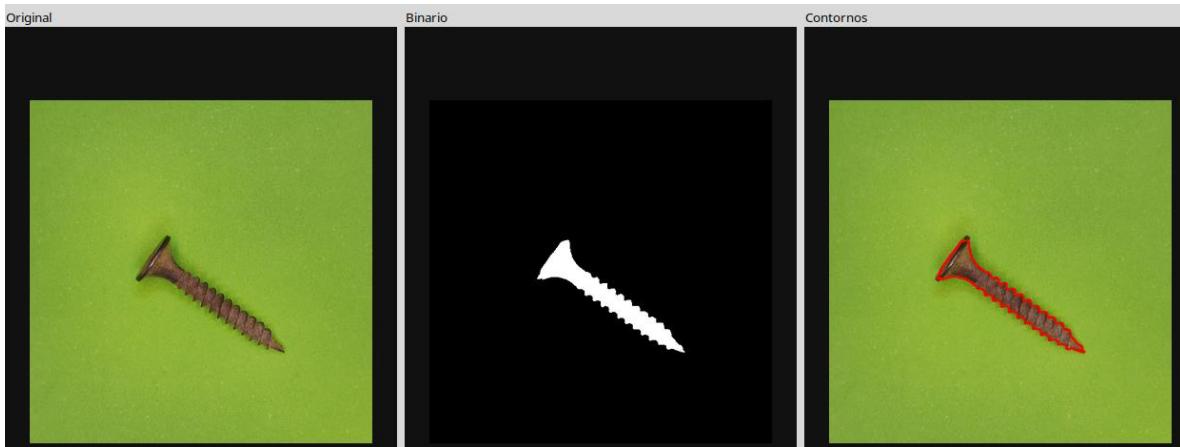


Aquí seleccionamos el directorio donde tenemos las imágenes a procesar, por defecto esta ruta apunta a “data/muestras”. Tenemos la opción de cambiar el directorio o de reconocer directamente todas las imágenes contenidas en la carpeta. Al realizar el reconocimiento aparecen todas las imágenes originales, con su nombre con extensión correspondiente, y luego la identificación de la pieza. Así como se muestra a continuación:



El reconocimiento procesa todas las imágenes, por temas de presentación aquí solo se muestran algunas del total de piezas procesadas.

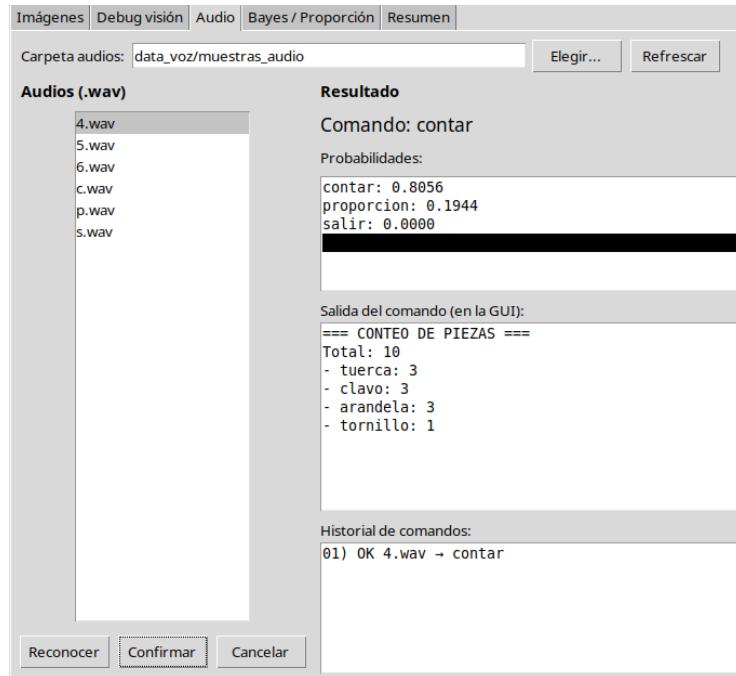
Si quisieramos ver los pasos intermedios, es decir, el preprocesamiento de la imagen, tenemos la pestaña “Debug Visión”. Aquí podemos observar el archivo original de la pieza, así como el binario, y los contornos hallados. También se da la opción de elegir cualquier otra imagen almacenada en el proyecto, solo se debe tener en cuenta que la imagen se halla tomado con determinadas condiciones (especificadas en la adquisición de imágenes).



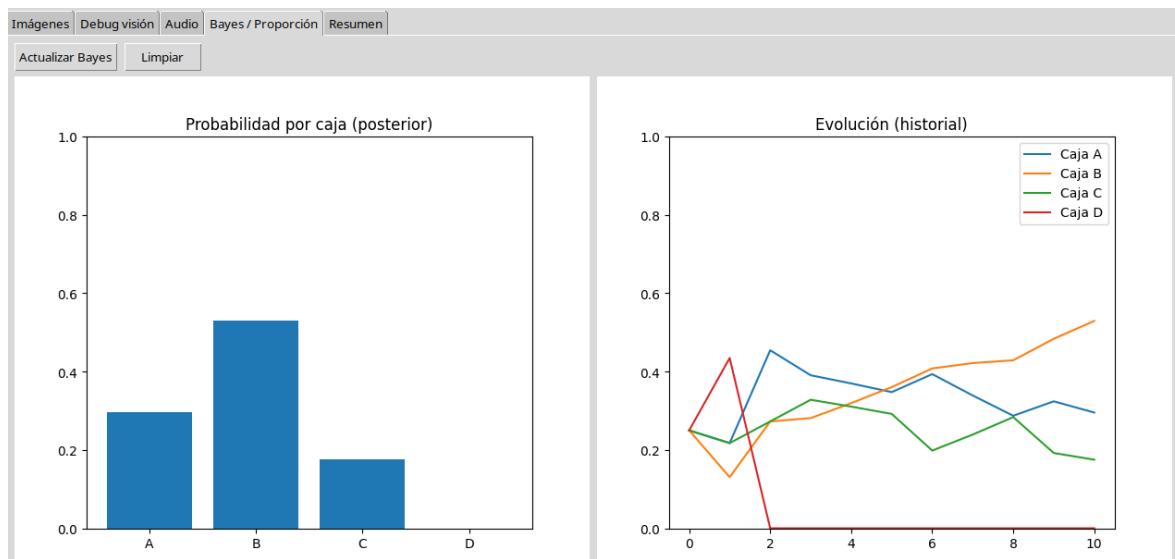
Luego en la pestaña audio tenemos la opción de elegir el directorio correspondiente. Al seleccionarlo podemos ver todos los archivos en formato WAV. Podemos elegir un único archivo por ver para reconocer. Los audios los nombramos con nombres que no indiquen nada acerca de su contenido.

Entonces una vez seleccionado el archivo, se procede a reconocer. Entonces aquí es donde el agente aplica todo el proceso detallado en el entrenamiento de KNN, se ejecutan todos los pasos correspondientes y se muestra la inferencia sobre el comando. Para poder ejecutar el comando identificado se debe seleccionar el botón confirmar. Esto se hace con el objetivo de poder reconocer diversos audios sin tener que ejecutarlos, y solo cuando se confirma se ejecuta el proceso indicado.

En este procesamos el primer audio del directorio, se reconoce el comando “contar”. Se confirma y se muestra el conteo de cada pieza del total de la muestra.



Luego podemos elegir otro archivo para procesar, en este caso procesamos el segundo e identificamos el comando “proporción”. Entonces se confirma el comando y se lleva a cabo el proceso. Esto muestra distribución de probabilidades aplicando el aprendizaje bayesiano, entonces esto nos lleva a la pestaña Bayes/Proporción.



Aquí se muestra en forma de grafico de barras la distribución de probabilidades de cada caja. Es notorio que la barra más alta indica cual es la caja de la cual es mas posible que se haya extraído la muestra de 10 piezas. En el otro gráfico, el de líneas, se muestra la evolución de las propiedades a lo largo del proceso de la extracción de cada pieza.

También tenemos una pestaña que muestra el resumen de todas las acciones ejecutadas en función de los comandos reconocidos. Para esta prueba que hemos hecho, el resumen de acciones de agente es el siguiente.

```

[Imágenes] [Debug visión] [Audio] [Bayes / Proporción] Resumen
[Generar resumen]

== RESUMEN ==
Total piezas: 10
- tuerca: 3
- clavo: 3
- arandela: 3
- tornillo: 1

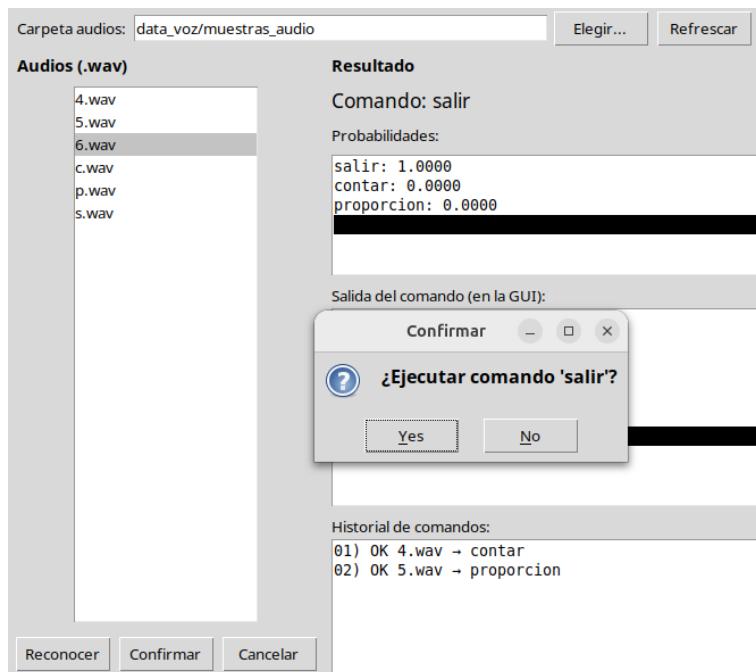
== BAYES ==
- Caja A: 0.2955
- Caja B: 0.4999
- Caja C: 0.1751
- Caja D: 0.0000
Caja más probable: B

== COMANDOS ==
01) EXECUTADO: contar (4.wav)
02) EXECUTADO: proporcion (5.wav)

== SALIDA / RESULTADOS ==
... Resultado 1 ...
== CONTEO DE PIEZAS ==
Total: 10
- tuerca: 3
- clavo: 3
- arandela: 3
- tornillo: 1

```

Cuando se reconoce el comando “salir”, al igual que con los otros comandos, primero debe confirmar antes de ejecutar. Esto finaliza el proceso y termina la ejecución.



RESULTADOS

En esta sección se presentan los resultados obtenidos por el agente desarrollado, analizando el desempeño de cada uno de los módulos que lo componen. Se detallan los datos utilizados, los conjuntos de prueba y las métricas empleadas para evaluar el comportamiento del sistema.

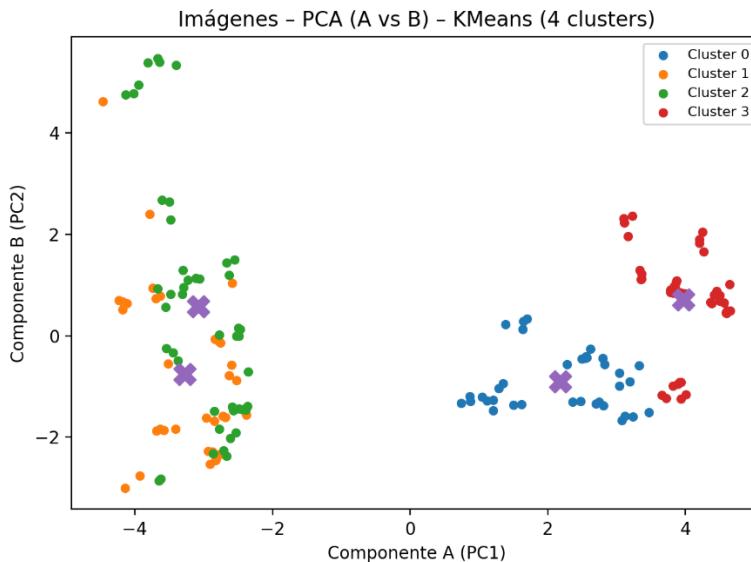
Resultados del módulo de visión artificial (K-Means)

Si bien K-Means es un algoritmo no supervisado, durante la etapa de evaluación se utilizó la etiqueta real únicamente para interpretar los clústeres generados y medir el desempeño del modelo.

Luego del entrenamiento, se construyó una matriz de confusión que compara la clase real de cada pieza con la clase asignada a partir del clúster más cercano. A partir de esta matriz se calcularon métricas de desempeño clásicas como precisión, recall y exactitud global. Los resultados muestran que el modelo alcanza una precisión global cercana al 94 %, lo que indica una alta capacidad de separación entre las clases en el espacio de características extraído.

El análisis de la matriz de confusión revela que las clases con formas bien diferenciadas, como las arandelas y las tuercas, presentan una tasa de clasificación prácticamente perfecta. En cambio, los errores observados se concentran principalmente entre clavos y tornillos, lo cual es esperable debido a su similitud geométrica, especialmente en imágenes donde la rosca del tornillo no es completamente visible o el clavo presenta irregularidades. A pesar de ello, la tasa de confusión se mantiene baja, confirmando que el conjunto de features seleccionados resulta adecuado para discriminar entre las distintas piezas.

Con el fin de mostrar los clusterers que se formaron se utilizó el método PCA para representar la nube de puntos que se forma en función a los features elegidos, y el resultado es el siguiente:



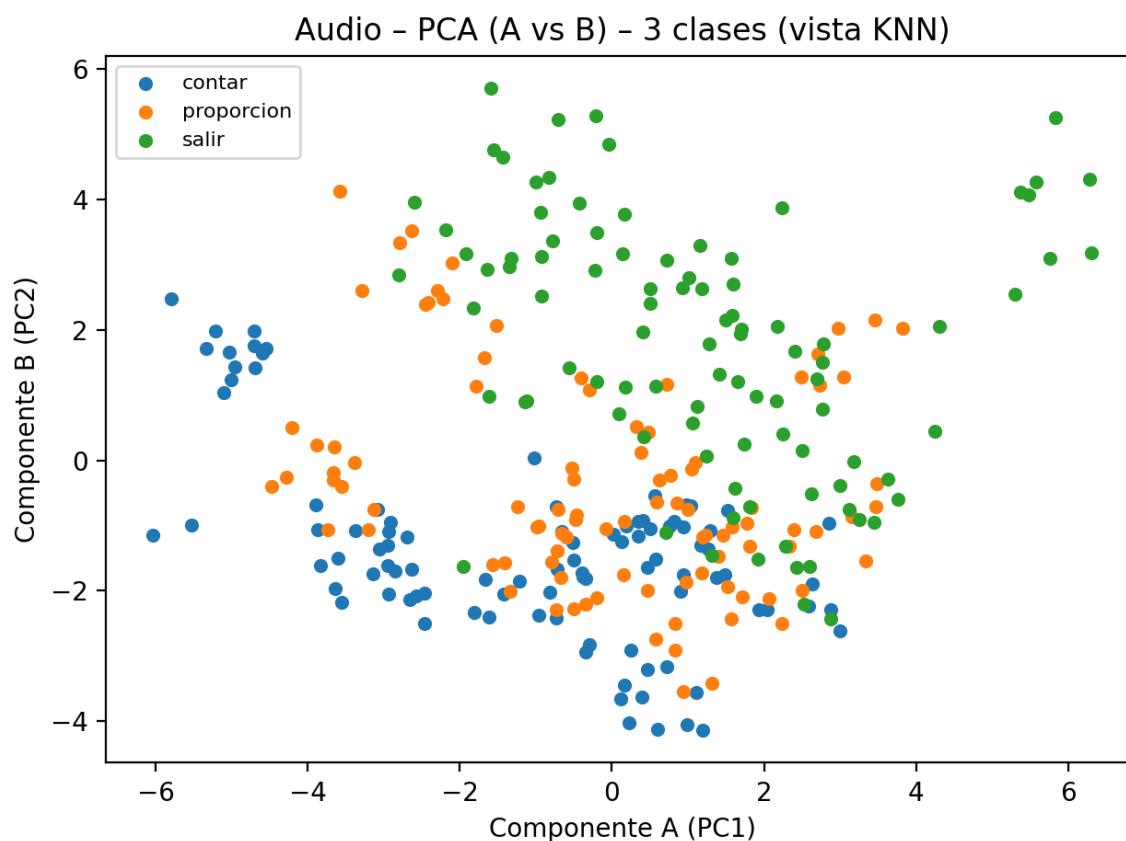
Estos resultados validan la elección de un enfoque no supervisado para la clasificación visual, demostrando que, aun sin utilizar etiquetas durante el aprendizaje, el agente es capaz de descubrir estructuras significativas en los datos y asociarlas correctamente a las clases reales.

Resultados del módulo de reconocimiento de voz (KNN)

Para el reconocimiento de comandos de voz se utilizó un clasificador KNN supervisado, entrenado a partir de un conjunto de audios correspondientes a los comandos posibles. Ya mencionamos que el dataset fue dividido en conjuntos de entrenamiento y prueba.

El modelo fue entrenado con un valor de $k=5$, utilizando distancia euclídea y ponderación inversa a la distancia, y se evaluó sobre el conjunto de prueba. La matriz de confusión obtenida muestra un alto nivel de coincidencia entre las clases reales y las predichas, con una exactitud global del 94,37 %. Los errores de clasificación observados corresponden mayormente a confusiones entre comandos con características acústicas similares o pronunciaciones cercanas, lo cual resulta coherente con la naturaleza del problema y con la variabilidad propia de la voz humana. No obstante, la baja tasa de error y la estabilidad del modelo confirman que el conjunto de características acústicas extraídas (MFCC, energía, centroides espectrales, entre otras) es suficiente para discriminar correctamente los comandos definidos.

Con el fin de complementar las métricas cuantitativas obtenidas en el reconocimiento de voz, se realizó una visualización del espacio de características mediante Análisis de Componentes Principales (PCA). Esta técnica permitió reducir los vectores originales de alta dimensión a dos componentes equivalentes (Componente A y Componente B), preservando las relaciones de cercanía entre las muestras.



Resultados del módulo bayesiano

El módulo de aprendizaje bayesiano no se evalúa mediante métricas usadas en los dos algoritmos anteriores, sino a través de la coherencia y estabilidad de la distribución de probabilidad obtenida para cada caja. A partir de las etiquetas de piezas clasificadas por el módulo de visión artificial, el agente realiza una actualización secuencial de las probabilidades utilizando el teorema de Bayes.

Los resultados muestran que, a medida que se incorporan observaciones, la probabilidad posterior converge progresivamente hacia una de las hipótesis, reduciendo la incertidumbre inicial. En los casos de prueba realizados, la caja de origen más probable coincidiría con la caja real de donde se extrae la muestra, lo que indica que el proceso de inferencia bayesiana integra de forma consistente la información proporcionada por el módulo de visión.

Además, la representación gráfica de la evolución de las probabilidades permite observar cómo cada nueva pieza clasificada refuerza o debilita las hipótesis existentes, brindando una interpretación más visual del razonamiento del agente.

Evaluación global del sistema

En conjunto, los resultados obtenidos demuestran que el agente cumple de manera satisfactoria con los objetivos planteados. El módulo de visión artificial logra una clasificación robusta de las piezas, el reconocimiento de voz presenta una alta precisión en la interpretación de comandos y el módulo bayesiano permite tomar decisiones probabilísticas coherentes bajo incertidumbre.

Si bien existen márgenes de mejora, particularmente en la discriminación entre piezas geométricamente similares y en la ampliación del dataset de audio, el desempeño alcanzado valida el enfoque adoptado y confirma que la integración de técnicas de aprendizaje supervisado, no supervisado y probabilístico constituye una solución eficaz para el problema planteado.

CONCLUSIONES

El desarrollo del agente inteligente presentado en este trabajo permitió integrar de manera funcional distintos conceptos de la Inteligencia Artificial, combinando aprendizaje no supervisado, aprendizaje supervisado y razonamiento probabilístico dentro de un mismo sistema. El objetivo principal, consistente en clasificar piezas metálicas mediante visión artificial, interpretar comandos de voz y estimar probabilísticamente la distribución de piezas en una caja de origen, fue alcanzado de forma satisfactoria según enfoque de la metodología propuesta.

Uno de los aspectos más relevantes del proyecto fue la importancia asignada a la correcta adquisición y construcción de las bases de datos, tanto de imágenes como de audios. A lo largo del desarrollo se comprobó que la calidad de los datos de entrada tiene un impacto directo en el desempeño global del agente. En el módulo de visión artificial, el diseño y construcción de un set de fotografía con iluminación controlada, fondo homogéneo y distancia constante permitió minimizar ruido, sombras y variaciones indeseadas, logrando imágenes consistentes que facilitaron enormemente las etapas posteriores de preprocesamiento, segmentación y extracción de características. De manera análoga, en el reconocimiento de voz, la definición de parámetros de grabación adecuados (formato WAV sin compresión, frecuencia de muestreo uniforme, audios mono y control de silencios) resultó clave para obtener señales comparables y estables, reduciendo errores atribuibles a factores externos.

Durante el desarrollo de cada etapa surgieron diversos problemas técnicos y conceptuales, que debieron resolverse de forma iterativa. En visión artificial, uno de los principales desafíos fue lograr una segmentación robusta que permitiera aislar correctamente las piezas independientemente de pequeñas variaciones de iluminación o posición. Este inconveniente se resolvió migrando del espacio de color BGR a HSV y definiendo rangos de color específicos para el fondo, complementados con operaciones morfológicas que limpian la máscara binaria sin deformar la geometría de los objetos. Otro problema relevante fue la correcta detección de agujeros internos en piezas como tuercas y arandelas, lo cual se solucionó utilizando detección de contornos jerárquica, permitiendo extraer características estructurales determinantes para la clasificación.

En el módulo de aprendizaje no supervisado con K-Means, se evidenció la necesidad de escalar y normalizar las características, dado que el algoritmo se basa en distancias euclídeas. Sin este paso, algunas variables dominaban el proceso de clustering y degradaban la separación entre clases. Asimismo, al tratarse de un algoritmo no supervisado, fue necesario implementar una estrategia posterior de etiquetado de clústeres mediante voto mayoritario, lo que permitió interpretar correctamente cada grupo sin alterar el proceso de aprendizaje. A pesar de estas dificultades, el modelo alcanzó una precisión aceptable, demostrando que el conjunto de descriptores geométricos y morfológicos seleccionados resulta adecuado para la clasificación de piezas.

En el reconocimiento de voz con KNN, los principales inconvenientes estuvieron asociados a la variabilidad de las voces, las diferencias de amplitud y la presencia de silencios. Estos problemas se resolvieron mediante un preprocesamiento cuidadoso de las señales. La correcta elección de características acústicas, como MFCC, energía y descriptores espectrales, permitió obtener un modelo robusto, con una exactitud global superior al 94 %.

El módulo de aprendizaje bayesiano permitió razonar explícitamente bajo incertidumbre, integrando información previa sobre la distribución de piezas en cada caja con la evidencia observada en la muestra. Los resultados obtenidos muestran una convergencia progresiva de las probabilidades hacia la caja correcta, lo que confirma la coherencia del enfoque probabilístico adoptado y su utilidad como mecanismo de decisión final.

Entre los puntos fuertes del agente implementado se destacan la modularidad del diseño, la correcta separación entre adquisición de datos, procesamiento, aprendizaje y la integración de distintos paradigmas de IA dentro de un mismo sistema funcional. La interfaz gráfica desarrollada aporta, asimismo, una visualización clara de los resultados y de las etapas intermedias, lo que mejora la interpretabilidad del comportamiento del agente.

Finalmente, como trabajos futuros, se propone ampliar y diversificar las bases de datos de imágenes y audios, incorporar técnicas más avanzadas de aprendizaje, como redes neuronales convolucionales para visión o modelos basados en deep learning para voz, y explorar la implementación del agente en un entorno físico real, integrándolo con actuadores como brazos robóticos o sistemas de clasificación automática. Estas mejoras permitirían aumentar la robustez, escalabilidad y aplicabilidad del sistema desarrollado.

En conclusión, el agente presentado cumple de manera eficiente con los objetivos planteados, demostrando que una correcta adquisición de datos, junto con una integración adecuada de técnicas de aprendizaje automático y razonamiento probabilístico, permite construir sistemas inteligentes robustos, interpretables y funcionales

BIBLIOGRAFÍA

- Poole, D. L., & Mackworth, A. K. (2017). *Artificial intelligence: Foundations of computational agents* (3rd ed.). Cambridge University Press.
- Russell, S. J., & Norvig, P. (2004). *Inteligencia artificial: Un enfoque moderno* (2^a ed.). Pearson Educación.
- Szeliski, R. (2022). *Computer vision: Algorithms and applications* (2nd ed.). Springer.
- Luna Casabene, J. I. (2025). Visión artificial [Apuntes de cátedra]. Inteligencia Artificial I, Universidad Champagnat.
- Rivera, S. S. (2025). Agentes inteligentes [Apuntes de cátedra]. Inteligencia Artificial I, Universidad Nacional de Cuyo.

IA GENERATIVA

En el desarrollo del presente trabajo se utilizó Inteligencia Artificial Generativa como herramienta de apoyo académico. En particular, se empleó el modelo ChatGPT, desarrollado por OpenAI, con el objetivo de asistir en:

- Redacción y corrección de secciones del informe,
- Clarificación de conceptos teóricos vinculados a Inteligencia Artificial y Visión Artificial,
- Organización del marco teórico,
- Elaboración de explicaciones sobre los algoritmos implementados (K-Means, KNN y clasificación),
- Formas de citar la bibliografía académica.