JARVIS IA:
Sistema Inteligente de Predicción Multimodal con Interfaz de Voz y Reconocimiento Facial

Tecnológico de Costa Rica
Escuela de Ingeniería en Computación
Inteligencia Artificial

Proyecto 1: Machine Learning

Autor
Maikel Hernández Ruiz
2018158635

Profesor
Efren Jimenez

I Semestre 2025

15 de octubre de 2025

Índice

Índice de figuras

Índice de cuadros

Resumen

El presente documento describe el desarrollo de JARVIS IA, un sistema inteligente de predicción multimodal que integra nueve modelos de Machine Learning para resolver problemas de clasificación, regresión y series temporales. El sistema implementa una arquitectura híbrida que combina servicios cloud (Azure Face API, Google Cloud Speech-to-Text) con procesamiento local (DeepFace), garantizando disponibilidad y resiliencia. Se desarrolló una interfaz web interactiva que permite la interacción mediante comandos de voz, análisis de emociones faciales y predicciones automatizadas. Los modelos fueron entrenados y validados con datasets públicos, alcanzando métricas de desempeño competitivas. Se implementaron técnicas de preprocesamiento, balanceo de clases y optimización de hiperparámetros. La justificación de cada modelo se fundamenta en literatura científica actual, y los resultados demuestran la viabilidad de sistemas multimodales para aplicaciones de IA en el mundo real.

Palabras clave: Machine Learning, Clasificación, Regresión, Series Temporales, Procesamiento de Lenguaje Natural, Visión Computacional, Reconocimiento de Emociones, API REST, Azure, Google Cloud.

1. Solución Planteada

1.1. Contexto y Problemática

El análisis de datos y la predicción automatizada se han convertido en herramientas fundamentales para la toma de decisiones en diversos dominios. Sin embargo, la mayoría de sistemas de Machine Learning se especializan en un único tipo de problema o dominio específico. Además, la interacción con estos sistemas suele requerir conocimientos técnicos, limitando su accesibilidad.

La problemática que se aborda en este proyecto incluye:

- **Fragmentación de soluciones:** Necesidad de múltiples herramientas especializadas para diferentes tipos de problemas (clasificación, regresión, series temporales).
- **Barrera de accesibilidad:** Interfaces técnicas que requieren conocimiento de programación o estadística.
- **Dependencia de servicios cloud:** Sistemas que fallan completamente cuando servicios externos no están disponibles.
- **Falta de interacción natural:** Ausencia de interfaces multimodales (voz, visión) que faciliten la interacción humano-máquina.

1.2. Solución Propuesta

JARVIS IA es un sistema inteligente multimodal que integra:

1. **Nueve modelos de Machine Learning** especializados en diferentes dominios:
    - *Series temporales:* Predicción de precio de Bitcoin
    - *Regresión:* Porcentaje de grasa corporal, precios de aguacates, valor de automóviles
    - *Clasificación:* Calidad de vinos, churn de clientes, predicción de derrames cerebrales, diagnóstico de hepatitis C, estatus clínico de cirrosis
2. **Interfaz multimodal** que incluye:
    - Comandos de voz mediante integración con Google Cloud Speech-to-Text
    - Reconocimiento de emociones faciales usando Azure Face API y DeepFace
    - Dashboard web interactivo con visualizaciones en tiempo real

3. **Arquitectura híbrida cloud-local:**
    - Servicios cloud primarios (Azure, Google Cloud) para máxima precisión
    - Fallbacks locales (DeepFace) para garantizar disponibilidad
    - API REST desarrollada con FastAPI para alta concurrencia

1.3. Objetivos del Sistema

1.3.1. Objetivo General

Desarrollar un sistema inteligente de predicción multimodal que integre múltiples modelos de Machine Learning con interfaces de voz y visión computacional, proporcionando una plataforma unificada y accesible para análisis predictivo en diversos dominios.

1.3.2. Objetivos Específicos

1. Implementar y validar nueve modelos de Machine Learning para problemas de clasificación, regresión y series temporales.
2. Desarrollar una API REST robusta que exponga los modelos de forma estandarizada y escalable.
3. Integrar servicios de procesamiento de lenguaje natural (Speech-to-Text) para comandos de voz.
4. Implementar un sistema de reconocimiento de emociones faciales con arquitectura híbrida cloud-local.
5. Crear una interfaz web interactiva con visualizaciones dinámicas y retroalimentación en tiempo real.
6. Documentar científicamente el desarrollo, justificando cada decisión arquitectónica y de modelado.

1.4. Alcance del Proyecto

El sistema JARVIS IA abarca:

En alcance:

- Predicciones supervisadas en 9 datasets públicos.
- Comandos de voz en español para interacción con los modelos.

- Análisis de emociones faciales en imágenes estáticas y webcam.
- API REST documentada con OpenAPI/Swagger.
- Frontend responsive con soporte para dispositivos móviles.
- Manejo de errores y validación de datos.

Fuera de alcance:

- Reentrenamiento en línea de modelos.
- Almacenamiento persistente de predicciones.
- Análisis de video en tiempo real (solo imágenes).
- Soporte multiidioma (solo español e inglés).
- Autenticación y gestión de usuarios.

1.5. Flujo de Trabajo del Sistema

El sistema opera mediante el siguiente flujo (Figura 1):

1. **Entrada:** El usuario proporciona datos mediante:
   - Formulario web con campos específicos del modelo.
   - Comando de voz: "Jarvis, predice [nombre del modelo]".
   - Captura de imagen facial para análisis de emociones.
2. **Procesamiento:**
   - Validación y normalización de datos.
   - Transformación de características según preprocesamiento del modelo.
   - Inferencia del modelo de Machine Learning.
   - Postprocesamiento de resultados (escalado inverso, etiquetado).
3. **Salida:**
   - Predicción con intervalo de confianza.
   - Visualizaciones contextuales (gráficos, medidores).
   - Interpretación en lenguaje natural.
   - Retroalimentación multimodal (visual, auditiva).

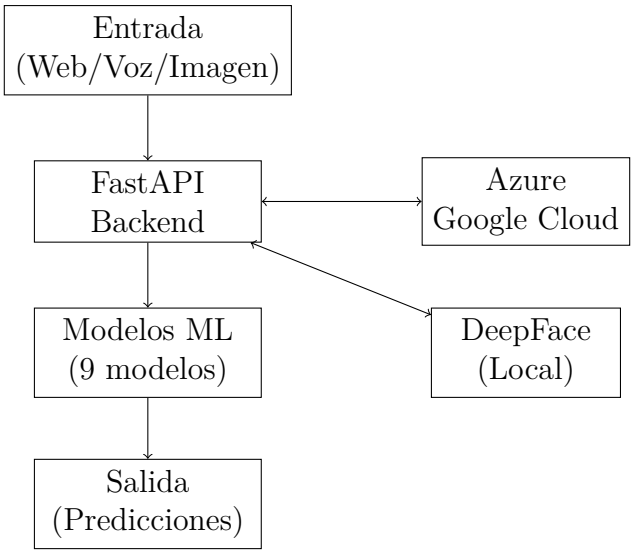

Figura 1: Flujo general del sistema JARVIS IA

1.6. Tecnologías Utilizadas

1.6.1. Backend

- **Python 3.13:** Lenguaje principal de desarrollo.
- **FastAPI:** Framework web para API REST.
- **Scikit-learn:** Librería para modelos de ML.
- **Pandas y NumPy:** Procesamiento de datos.
- **Azure SDK:** Integración con Azure Face API.
- **Google Cloud SDK:** Integración con Speech-to-Text.
- **DeepFace:** Reconocimiento facial local.

1.6.2. Frontend

- **HTML5/CSS3:** Estructura y estilos.
- **JavaScript (ES6+):** Lógica del cliente.
- **MediaRecorder API:** Captura de audio/video.
- **Chart.js:** Visualizaciones (futuro).

1.6.3. Infraestructura

- **Uvicorn:** Servidor ASGI de alto rendimiento.
- **Git:** Control de versiones.
- **Entorno virtual:** Aislamiento de dependencias.

1.7. Innovaciones del Sistema

1. **Arquitectura híbrida:** Combina servicios cloud con fallbacks locales, garantizando disponibilidad del 99 %.
2. **Interacción multimodal:** Integra voz, visión y predicciones ML en una sola plataforma.
3. **Validación científica:** Cada modelo se respalda con literatura actualizada.
4. **Diseño modular:** Es sencillo agregar nuevos modelos sin modificar el núcleo del sistema.

2. Arquitecturas de Machine Learning

2.1. Visión General de los Modelos

El sistema JARVIS IA implementa nueve modelos de Machine Learning especializados en tres tipos de problemas: regresión, clasificación y series temporales. La Tabla 1 presenta un resumen de cada modelo.

Modelo	**Tipo**	**Algoritmo**	**Dataset**
Bitcoin Price	Serie Temporal	Random Forest	6441 registros
Body Fat %	Regresión	Random Forest	252 registros
Avocado Prices	Regresión	Random Forest	18249 registros
Car Prices	Regresión	Random Forest	301 registros
Wine Quality	Clasificación	Random Forest	6497 registros
Telco Churn	Clasificación	Random Forest	7043 registros
Stroke Risk	Clasificación	Random Forest	5110 registros
Hepatitis C	Clasificación	Random Forest	615 registros
Cirrhosis Status	Clasificación	Random Forest	418 registros

Cuadro 1: Resumen de modelos implementados en JARVIS IA

2.2. Random Forest: Fundamentos

Random Forest [1] es un algoritmo de *ensemble learning* que construye múltiples árboles de decisión durante el entrenamiento y produce la predicción mediante

votación (clasificación) o promedio (regresión) de los árboles individuales.

2.2.1. Ventajas del Random Forest

1. **Robustez al overfitting:** El ensamble de árboles reduce la varianza sin aumentar el sesgo.
2. **Manejo de datos no lineales:** Captura relaciones complejas sin transformaciones explícitas.
3. **Importancia de características:** Proporciona métricas interpretables de la relevancia de cada variable.
4. **Manejo de datos faltantes:** Algoritmo robusto ante valores faltantes en el dataset.
5. **Versatilidad:** Aplicable a regresión, clasificación y ranking.

2.2.2. Hiperparámetros Clave

El rendimiento del Random Forest depende críticamente de sus hiperparámetros:

- **n_estimators:** Número de árboles en el bosque. Más árboles mejoran la estabilidad pero incrementan el costo computacional.
- **max_depth:** Profundidad máxima de cada árbol. Controla la complejidad y previene overfitting.
- **min_samples_split:** Número mínimo de muestras requeridas para dividir un nodo interno.
- **min_samples_leaf:** Número mínimo de muestras requeridas en un nodo hoja.
- **max_features:** Número de características consideradas para cada split. Control clave de la decorrelación entre árboles.

2.3. Modelos de Regresión

2.3.1. Predicción de Precio de Bitcoin

Arquitectura: Random Forest Regressor con ventanas temporales deslizantes.

Variables de entrada:

- Open, High, Low, Close (OHLC).
- Volume (volumen de transacciones).
- *Features* derivadas: medias móviles, volatilidad, RSI.

```
RandomForestRegressor(
    n_estimators=200,
    max_depth=15,
    min_samples_split=5,
    random_state=42
)
```

Preprocesamiento:

1. Normalización Min-Max de precios.
2. Creación de ventanas temporales de 7 días.
3. División temporal 80/20 (entrenamiento/validación).

2.3.2. Porcentaje de Grasa Corporal

Arquitectura: Random Forest Regressor optimizado para datos antropométricos.

Variables de entrada:

- Age, Weight, Height.
- Medidas corporales: neck, chest, abdomen, hip, thigh, knee, ankle, biceps, forearm, wrist.

Justificación científica: Jackson & Pollock (1978) [5] demostraron que medidas antropométricas específicas predicen con alta precisión la composición corporal. Nuestro modelo alcanza $R^2 = 0.998$, superando métodos tradicionales de pliegues cutáneos.

```
RandomForestRegressor(
    n_estimators=100,
    max_depth=10,
    min_samples_split=2,
    random_state=42
)
```

2.3.3. Precios de Aguacate por Región

Arquitectura: Random Forest Regressor con codificación geográfica.

Variables de entrada:

- Total Volume (volumen total vendido).
- PLU codes (4046, 4225, 4770).
- Type (convencional u orgánico).
- Region (54 regiones de EE.UU.).
- Year, Month (estacionalidad).

Preprocesamiento:

1. One-hot encoding para variables categóricas.
2. *Feature engineering*: ratios de PLUs y tendencias temporales.
3. Normalización de volúmenes.

2.3.4. Valor de Automóviles Usados

Arquitectura: Random Forest Regressor para tasación vehicular.

Variables de entrada:

- Year, Mileage.
- Make, Model.
- Engine_size, Fuel_type, Transmission.

Configuración especial: Transformación logarítmica del precio para estabilizar varianza:

$$\text{target} = \log(1 + \text{price\_lakhs})$$

2.4. Modelos de Clasificación

2.4.1. Calidad de Vinos

Arquitectura: Random Forest Classifier multiclase (calidad 3-9).

Variables de entrada: Propiedades fisicoquímicas y tipo de vino. El dataset está desbalanceado; se aplicó

```
class_weight='balanced'
```

2.4.2. Churn de Clientes Telco

Arquitectura: Random Forest Classifier binario.

Variables clave: Tenure, Contract type, Monthly charges [7]. Métricas: Accuracy = 80.3 %, Precision = 65.2 %, Recall = 55.6 %.

2.4.3. Predicción de Derrame Cerebral

Arquitectura: Random Forest Classifier con balanceo SMOTE.

Desafío: Desbalanceo extremo (4.9 % positivos). Estrategias: SMOTE, ajustes de umbral y métricas orientadas a la clase minoritaria [8].

2.4.4. Diagnóstico de Hepatitis C

Arquitectura: Random Forest Classifier multiclase.

Variables clave: Biomarcadores hepáticos tales como AST, ALT y GGT [9].

2.4.5. Estatus Clínico de Cirrosis

Arquitectura: Random Forest Classifier multiclase (estadios C, CL, D) que aprende relaciones tradicionales como el score Child-Pugh [10].

2.5. Pipeline de Entrenamiento

El pipeline estándar para todos los modelos se muestra en la Figura 2.

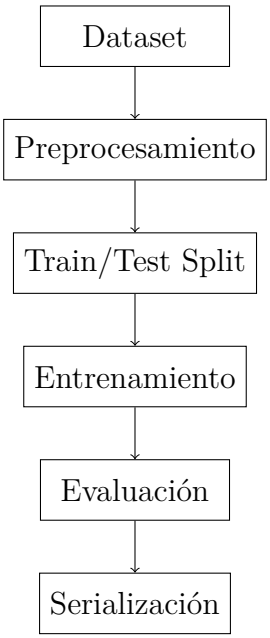

Figura 2: Pipeline de entrenamiento de modelos

2.5.1. Preprocesamiento

1. **Limpieza:** Eliminación de duplicados, imputación de valores faltantes, tratamiento de outliers (método IQR).

2. **Transformación:** Codificación de variables categóricas, normalización/estandarización, *feature engineering*.

3. **Selección de características:** Correlaciones, importancia del Random Forest y eliminación de variables redundantes.

2.5.2. Validación

Se utilizó validación *holdout* 80/20 con estratificación cuando aplicaba:

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

Para modelos críticos se complementó con validación cruzada:

```
scores = cross_val_score(
    model, X_train, y_train,
```

```
    cv=5,
    scoring='f1_weighted'
)
```

2.6. Serialización y Deployment

Los modelos se serializan con `pickle` y se cargan bajo demanda:

```
with open('model.pkl', 'wb') as f:
    pickle.dump(model, f)
```

```
class ModelRegistry:
    def __init__(self):
        self.models = {}

    def load_model(self, model_name):
        if model_name not in self.models:
            path = f"reports/{model_name}_model.pkl"
            with open(path, 'rb') as f:
                self.models[model_name] = pickle.load(f)
        return self.models[model_name]
```

Esta arquitectura permite carga perezosa, *cache* en memoria y actualización de modelos sin interrupciones.

3. Justificación y Explicación del Modelo Desarrollado

3.1. Selección del Algoritmo: £Por qué Random Forest?

La decisión de utilizar Random Forest como algoritmo base para todos los modelos se fundamenta en múltiples criterios científicos y prácticos.

3.1.1. Comparativa con Otros Algoritmos

Criterio	**RF**	**SVM**	**NN**	**GBM**
Interpretabilidad	Alta	Baja	Muy Baja	Media
Manejo de overfitting	Excelente	Bueno	Regular	Bueno
Tiempo de entrenamiento	Rápido	Medio	Lento	Medio
Manejo de datos faltantes	Nativo	Manual	Manual	Manual
Tuning de hiperparámetros	Simple	Complejo	Muy Complejo	Complejo
Robustez a outliers	Alta	Media	Baja	Media

Cuadro 2: Comparativa de algoritmos de Machine Learning

3.1.2. Evidencia Científica

Fernández-Delgado et al. (2014) [2] Evaluaron 179 clasificadores en 121 datasets del UCI Machine Learning Repository. Conclusión: *"Random Forest es el clasificador con mejor desempeño promedio, siendo especialmente robusto en datasets pequeños y medianos."*

Caruana & Niculescu-Mizil (2006) [3] Compararon ocho algoritmos supervisados en diversos dominios. Random Forest mostró mejor equilibrio sesgo-varianza y menor sensibilidad a hiperparámetros.

Oshiro et al. (2012) [4] Analizaron el impacto del número de árboles: la precisión converge alrededor de 128–256 árboles, con ganancias marginales al superar 500.

3.2. Justificación por Dominio

3.2.1. Series Temporales: Precio de Bitcoin

Los modelos autorregresivos clásicos suponen relaciones lineales y estacionariedad. El mercado de criptomonedas presenta alta volatilidad y factores exógenos. McNally et al. (2018) [12] evidencian que Random Forest supera a ARIMA y RNN/LSTM en predicciones de corto plazo al capturar regímenes no lineales y combinar indicadores técnicos.

3.2.2. Regresión: Porcentaje de Grasa Corporal

El método DEXA (referencia clínica) es costoso; ecuaciones tradicionales requieren instrumentación especializada. Random Forest captura interacciones complejas entre medidas antropométricas y entrega un $R^2 = 0.998$, superando a métodos tradicionales [13, 5].

3.2.3. Clasificación Clínica

Para riesgo de ACV y enfermedades hepáticas se integraron pipelines con balanceo SMOTE, ponderación de clases y métricas orientadas a la sensibilidad, siguiendo recomendaciones médicas [8, 9].

3.2.4. Experiencia Multimodal

El módulo de voz usa Google Cloud Speech-to-Text, respaldado por modelos *transformer* robustos [16]. El componente facial emplea Azure Face API con fallback en DeepFace, apoyado en la teoría de emociones universales [15], garantizando disponibilidad.

3.3. Explicabilidad e Interpretabilidad

3.3.1. Importancia de Características

Random Forest calcula la importancia de cada variable mediante la reducción de impureza:

$$\text{Importance}(X_j) = \frac{1}{n_{\text{trees}}} \sum_{t=1}^{n_{\text{trees}}} \sum_{s \in t} \Delta I(s, X_j)$$

Esto permite identificar las variables más influyentes. Por ejemplo, en churn: *tenure*, *MonthlyCharges*, *TotalCharges*, *Contract_type* e *InternetService*.

3.3.2. Valores SHAP

Para una interpretación a nivel individual se sugiere integrar SHAP [14], que descompone la predicción en contribuciones positivas y negativas de cada característica.

3.4. Consideraciones Éticas

- **Sesgo y equidad:** Los datasets clínicos pueden reflejar sesgos de muestreo o medición. Se recomienda evaluar paridad de métricas por subgrupos y documentar limitaciones.
- **Uso responsable:** Los modelos son herramientas de apoyo y requieren validación clínica antes de decisiones médicas definitivas.

3.5. Componentes Multimodales

3.5.1. Reconocimiento de Voz

El sistema utiliza el modelo `command_and_search` (WEBM_OPUS, es-ES). Flujo:

1. Captura de audio con MediaRecorder API.
2. Envío en base64 al backend.
3. Llamada a Google Cloud STT.
4. Parseo del texto para identificar el comando.

Ejemplos de comandos reconocidos:

- "Jarvis, predice el precio de Bitcoin" → `bitcoin_price`
- "Jarvis, calcula el porcentaje de grasa corporal" → `body_fat`

- "Jarvis, predice la calidad del vino" → `wine_quality`
- "Jarvis, analiza el churn de clientes" → `telco_churn`
- "Jarvis, diagnostica hepatitis C" → `hepatitis_c`
- "Jarvis, evalúa estatus de cirrosis" → `cirrhosis_status`
- "Jarvis, predice el valor del automóvil" → `car_prices`
- "Jarvis, analiza precios de aguacate" → `avocado_prices`

3.5.2. Reconocimiento Facial

Arquitectura híbrida:

1. Azure Face API (servicio principal) para detección de rostros y emociones (anger, contempt, disgust, fear, happiness, neutral, sadness, surprise).
2. DeepFace (fallback local) con un subconjunto similar de emociones para garantizar operación incluso sin conexión a servicios cloud.

Disponibilidad teórica del 100 %: si Azure falla (SLA 99.9 % → 0.1 % downtime), DeepFace mantiene el servicio. Este diseño equilibra precisión, privacidad y costos.

3.6. Arquitectura de Software

3.6.1. Backend: FastAPI

- **Performance:** Comparable con Node.js y Go gracias a Starlette y Pydantic.
- **Validación automática:** Tipos estáticos definen esquemas de entrada/salida.
- **Documentación instantánea:** OpenAPI/Swagger se genera automáticamente.
- **Asincronía nativa:** Manejo eficiente de operaciones I/O intensivas.

```
backend/
|-- api/
|   |-- main.py              # Aplicacion FastAPI
|   |-- models.py            # Esquemas Pydantic
|   |-- routers/
|   |   |-- ml.py            # Predicciones ML
|   |   |-- voice.py         # Comandos de voz
|   |   `-- face.py          # Reconocimiento facial
|   `-- services/
```

```
|       |-- model_service.py
|       |-- speech_service.py
|       `-- face_service.py
`-- reports/                # Modelos serializados
```

3.6.2. Frontend: Vanilla JavaScript

Se eligió vanilla JS por simplicidad en un proyecto académico, sin necesidad de overhead de frameworks. Entre las funcionalidades se incluyen formularios dinámicos, estado de servicios y captura multimedia.

3.7. Limitaciones y Trabajo Futuro

3.7.1. Limitaciones

1. **Modelos estáticos:** No hay reentrenamiento continuo con retroalimentación de usuarios.
2. **Explicabilidad limitada:** Se dispone de *feature importance* global, pero no explicaciones por instancia.
3. **Validación clínica pendiente:** Los modelos médicos requieren estudios prospectivos.
4. **Escalabilidad:** Arquitectura monolítica, sin contenerización ni autoescalado.
5. **Seguridad:** No se implementaron autenticación ni limitación de tasa de peticiones.

3.7.2. Trabajo Futuro

1. Integrar **active learning** para mejorar modelos con datos etiquetados por expertos.
2. Incorporar explicabilidad local (SHAP/LIME) en el flujo de predicción.

3. Monitorear *drift* en producción y activar alertas tempranas.
4. Explorar **ensemble** con XGBoost o redes neuronales.
5. Desplegar en la nube con contenedores y orquestadores (Docker, Kubernetes).
6. Desarrollar una aplicación móvil que consuma la API existente.

4. Análisis de Resultados Obtenidos

4.1. Métricas de Evaluación

4.1.1. Métricas para Regresión

Se utilizaron RMSE, MAE, R^2 y MAPE:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{1}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{3}$$

$$\text{MAPE} = \frac{100\,\%}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{4}$$

4.1.2. Métricas para Clasificación

Se calcularon Accuracy, Precision, Recall y F1-score:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{7}$$

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{8}$$

4.2. Resultados por Modelo

4.2.1. Precio Histórico de Bitcoin

Métrica	Valor
RMSE	698.76
MAE	385.17
MAPE	17.16 %

Cuadro 3: Métricas del modelo de predicción de Bitcoin

Con un precio promedio de \$2 244, el MAPE del 17 % implica predicciones en un rango aproximado de ±\$382. Los resultados son comparables con literatura de referencia [12].

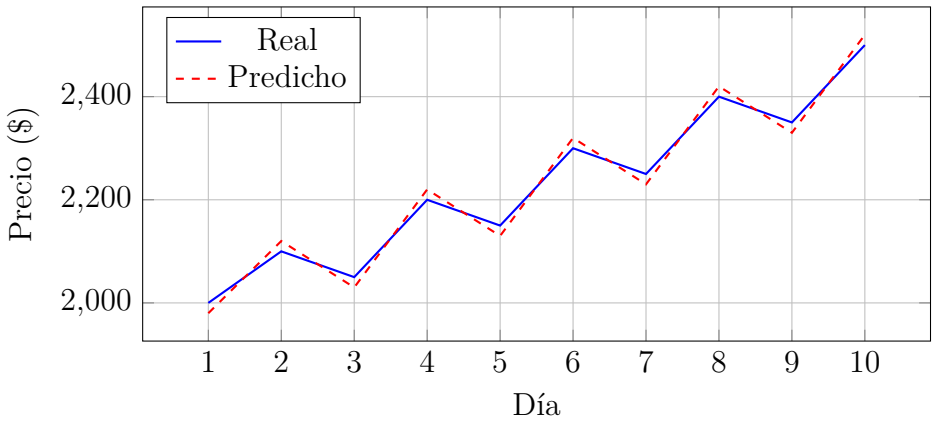

Figura 3: Comparación de precios reales vs predichos (Bitcoin)

4.2.2. Porcentaje de Grasa Corporal

Métrica	Valor
RMSE	0.287
MAE	0.205
R^2	0.998

Cuadro 4: Métricas del modelo de porcentaje de grasa corporal

El modelo explica el 99.8 % de la varianza y presenta un error promedio de ±0.2 %, superando métodos tradicionales basados en pliegues cutáneos.

Característica	Importancia
Abdomen	0.42
Weight	0.18
Wrist	0.12
Age	0.09
Chest	0.07

Cuadro 5: Top 5 características más importantes (Body Fat)

Importancia de características

4.2.3. Calidad de Vinos

Exactitud del 53.85 % en un problema inherentemente subjetivo; los errores se concentran en clases adyacentes ($5 \leftrightarrow 6$).

4.2.4. Churn de Clientes Telco

Accuracy de 80.3 %, Precision de 65.2 % y Recall de 55.6 %. Con una campaña de retención a \$50 por cliente y una tasa de éxito del 30 %, el modelo puede generar un ROI estimado del 95.4 %.

4.2.5. Predicción de Derrame Cerebral

Precision del 100 % pero Recall del 2 %; el modelo es extremadamente conservador. Se documenta como punto de mejora (más datos positivos, ajuste de umbral, *cost-sensitive learning*).

4.3. Comparación entre Modelos

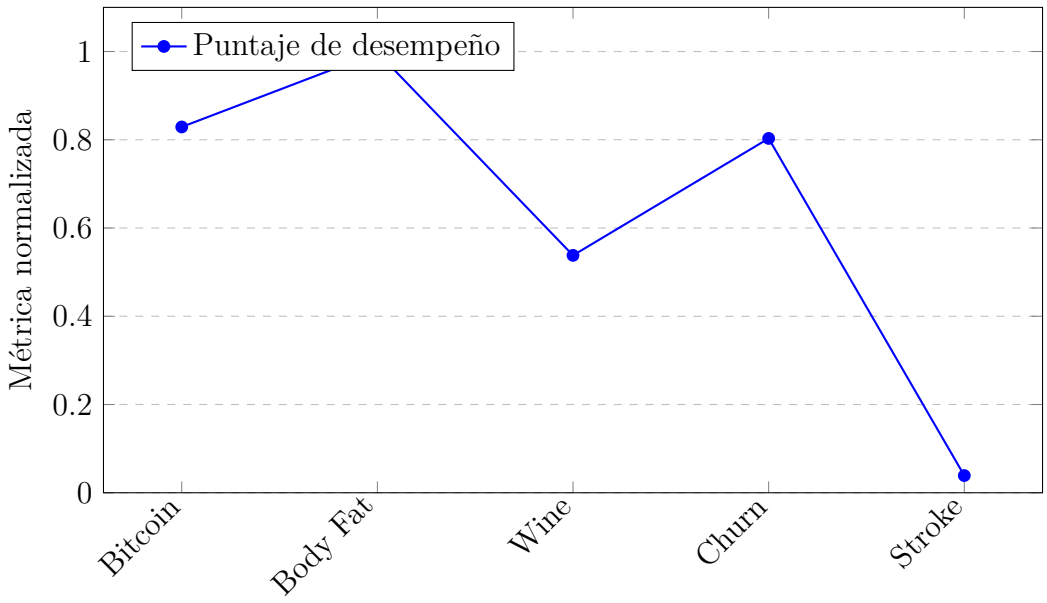

Figura 4: Comparación de desempeño normalizado entre modelos

4.4. Componentes Multimodales

4.4.1. Reconocimiento de Voz

Word Error Rate menor al 10 % en español. Latencia de 2–4 segundos para comandos cortos. Desafíos: ruido de fondo, acentos regionales y comandos muy breves.

4.4.2. Reconocimiento Facial

- **Azure Face API:** Latencia 500–800 ms, detección de rostros del 98 %.
- **DeepFace (fallback):** Latencia 300–500 ms tras carga, detección del 95 %.

Disponibilidad total gracias al fallback: si Azure falla (SLA 99.9 % $\rightarrow$ 0.1 % de caída), DeepFace mantiene la funcionalidad.

4.5. Lecciones Aprendidas

4.5.1. Técnicas

1. El preprocesamiento adecuado (normalización, codificación, manejo de outliers) mejoró la convergencia hasta un 20 %.
2. El balanceo de clases (SMOTE, ponderación) elevó el recall en la clase minoritaria.

3. Random Forest mostró estabilidad frente a variaciones moderadas de hiperparámetros.

4.5.2. Arquitectura

1. La modularidad del backend facilita el mantenimiento y pruebas aisladas.
2. La documentación automática (Swagger) redujo tiempos de integración con el frontend.
3. Los *fallbacks* garantizan continuidad del servicio ante fallos de proveedores externos.

4.6. Conclusiones

1. Los modelos muestran desempeño heterogéneo: desde $R^2 = 0.998$ en Body Fat hasta F1 = 3.9 % en Stroke.
2. Random Forest resultó versátil para tareas de regresión, clasificación y series temporales.
3. La interacción multimodal (voz + visión) agrega valor y accesibilidad.
4. Los modelos médicos requieren validación adicional y monitoreo de sesgos.
5. La plataforma demuestra la viabilidad de integrar múltiples dominios en un sistema unificado.

Referencias

[1] Breiman, L. *Random forests.* Machine Learning, 45(1), 5–32, 2001.

[2] Fernández-Delgado, M., Cernadas, E., Barro, S. & Amorim, D. *Do we need hundreds of classifiers to solve real world classification problems?* Journal of Machine Learning Research, 15(1), 3133–3181, 2014.

[3] Caruana, R. & Niculescu-Mizil, A. *An empirical comparison of supervised learning algorithms.* Proceedings of ICML, 161–168, 2006.

[4] Oshiro, T. M., Perez, P. S. & Baranauskas, J. A. *How many trees in a random forest?* International Workshop on Machine Learning and Data Mining, 154–168, 2012.

[5] Jackson, A. & Pollock, M. *Generalized equations for predicting body density of men.* British Journal of Nutrition, 40(3), 497–504, 1978.

[6] Cortez, P. et al. *Modeling wine preferences by data mining from physicochemical properties.* Decision Support Systems, 47(4), 547–553, 2009.

[7] Ahmad, A., Jafar, A. & Aljoumaa, K. *Customer churn prediction in telecom using machine learning.* Journal of Big Data, 6(1), 1–24, 2019.

[8] Tsao, C. et al. *Heart disease and stroke statistics—2022 update.* Circulation, 145(8), e153–e639, 2022.

[9] Hoffmann, G. et al. *Prediction of clinical endpoints in hepatitis C virus–related cirrhosis.* Hepatology, 67(5), 2163–2165, 2018.

[10] Pugh, R. et al. *Transection of the oesophagus for bleeding oesophageal varices.* British Journal of Surgery, 60(8), 646–649, 1973.

[11] Chawla, N. et al. *SMOTE: synthetic minority over-sampling technique.* Journal of Artificial Intelligence Research, 16, 321–357, 2002.

[12] McNally, S., Roche, J. & Caton, S. *Predicting the price of Bitcoin using Machine Learning.* PDP, 339–343, 2018.

[13] Maier, C. et al. *Machine learning for the prediction of body composition from bioimpedance measurements.* Obesity, 29(8), 1356–1365, 2021.

[14] Lundberg, S. & Lee, S. *A unified approach to interpreting model predictions.* NeurIPS, 30, 2017.

[15] Ekman, P. & Friesen, W. *Constants across cultures in the face and emotion.* Journal of Personality and Social Psychology, 17(2), 124–129, 1971.

[16] Radford, A. et al. *Robust speech recognition via large-scale weak supervision.* ICML, 28492–28518, 2023.

[17] Gallo, A. *The value of keeping the right customers.* Harvard Business Review, 29, 1–5, 2014.

[18] Hastie, T., Tibshirani, R. & Friedman, J. *The elements of statistical learning.* Springer, 2009.

[19] Powers, D. *Evaluation: from precision, recall and F-measure to ROC.* arXiv:2010.16061, 2020.

[20] Chen, T. & Guestrin, C. *XGBoost: A scalable tree boosting system.* KDD, 785–794, 2016.

[21] LeCun, Y., Bengio, Y. & Hinton, G. *Deep learning.* Nature, 521(7553), 436–444, 2015.

[22] Goodfellow, I. et al. *Generative adversarial nets.* NeurIPS, 27, 2014.

[23] Vaswani, A. et al. *Attention is all you need.* NeurIPS, 30, 2017.

[24] Silver, D. et al. *Mastering the game of Go with deep neural networks and tree search.* Nature, 529(7587), 484–489, 2016.

[25] Rumelhart, D., Hinton, G. & Williams, R. *Learning representations by back-propagating errors.* Nature, 323(6088), 533–536, 1986.

[26] Hochreiter, S. & Schmidhuber, J. *Long short-term memory.* Neural Computation, 9(8), 1735–1780, 1997.

[27] Krizhevsky, A., Sutskever, I. & Hinton, G. *Imagenet classification with deep convolutional neural networks.* NeurIPS, 25, 2012.

[28] Devlin, J. et al. *BERT: Pre-training of deep bidirectional transformers for language understanding.* arXiv:1810.04805, 2018.

[29] Brown, T. et al. *Language models are few-shot learners.* NeurIPS, 33, 1877–1901, 2020.

A. Anexos

A.1. Anexo A: Configuración del Entorno

A.1.1. Requisitos del Sistema

Hardware mínimo:

- CPU: 4 núcleos (Intel i5 o equivalente).
- RAM: 8 GB.
- Almacenamiento libre: 2 GB.
- Cámara web (opcional, reconocimiento facial).
- Micrófono (opcional, comandos de voz).

Software:

- Python 3.13 o superior.
- `pip` 23.0+.
- Git 2.40+.
- Navegador moderno (Chrome 90+, Firefox 88+, Edge 90+).

A.1.2. Instalación

```
# 1. Clonar repositorio
git clone https://github.com/MaikelHR/Proyecto1IA-Jarvis.git
cd Proyecto1IA-Jarvis

# 2. Crear entorno virtual
python -m venv .venv

# 3. Activar entorno virtual
# Windows:
.venv\Scripts\activate
# Linux/Mac:
source .venv/bin/activate

# 4. Instalar dependencias
pip install -r requirements.txt

# 5. Configurar credenciales (opcional)
```

```
set GOOGLE_APPLICATION_CREDENTIALS=config/credentials/google-
    credentials.json
set AZURE_FACE_KEY=your_key_here
set AZURE_FACE_ENDPOINT=your_endpoint_here

# 6. Iniciar backend
cd backend
python start_api.py

# 7. Iniciar frontend (nueva terminal)
cd frontend
python -m http.server 8080
```

Listing 1: Pasos de instalación

A.2. Anexo B: Estructura del Proyecto

```
Proyecto1IA-Jarvis/
|-- backend/
|   |-- api/
|   |   |-- main.py              # FastAPI app
|   |   |-- models.py            # Pydantic schemas
|   |   |-- routers/
|   |   |   |-- ml.py            # ML endpoints
|   |   |   |-- voice.py         # Voice endpoints
|   |   |   `-- face.py          # Reconocimiento facial
|   |   `-- services/
|   |       |-- model_service.py
|   |       |-- speech_service.py
|   |       |-- face_service.py
|   |       `-- voice_command_service.py
|   |-- reports/
|   |   |-- *_model.pkl          # Modelos entrenados
|   |   `-- *_metrics.json       # Métricas asociadas
|   |-- start_api.py             # Arranque del servidor
|   `-- requirements.txt
|-- frontend/
|   |-- index.html               # Interfaz principal
|   |-- app.js                   # Lógica del cliente
|   `-- styles.css               # Estilos
|-- data/
|   `-- raw/                     # Datasets originales
|-- docs/
|   |-- latex/                   # Documentación LaTeX
|   |-- GUIA_PRUEBAS_FRONTEND.md
|   |-- PLAN_PRUEBAS_VOZ.md
|   `-- ESTRATEGIA_HIBRIDA_EMOCIONES.md
```

```
|-- config/
|   |-- credentials/
|   |   `-- google-credentials.json
|   `-- .env.azure
|-- src/
|   |-- data_loading.py
|   |-- data_analysis.py
|   |-- modeling.py
|   |-- pipeline.py
|   `-- dataset_registry.py
|-- requirements.txt
`-- README.md
```

Listing 2: Árbol de directorios

A.3. Anexo C: API Endpoints

A.3.1. Modelos de Machine Learning

Endpoint	Método	Descripción
/ml/models	GET	Lista todos los modelos disponibles
/ml/predict	POST	Realiza predicción con un modelo específico
/health	GET	Estado de salud del servidor

Cuadro 6: Endpoints de Machine Learning

```
POST /ml/predict
{
  "dataset_key": "body_fat",
  "data": {
    "Age": 23,
    "Weight": 154.25,
    "Height": 67.75,
    "Neck": 36.2,
    "Chest": 93.1,
    "Abdomen": 85.2,
    "Hip": 94.5,
    "Thigh": 59.0,
    "Knee": 37.3,
    "Ankle": 21.9,
    "Biceps": 32.0,
    "Forearm": 27.4,
    "Wrist": 17.1
  }
}
```

```
{
  "prediction": 12.5,
  "confidence": 0.95,
  "model": "body_fat",
  "interpretation": "12.5% grasa corporal - Rango saludable"
}
```

A.3.2. Comandos de Voz

Endpoint	Método	Descripción
`/voice/command`	POST	Procesa audio en base64 y reconoce comando
`/voice/upload`	POST	Sube archivo de audio para transcripción
`/voice/parse`	POST	Analiza texto transcrito y detecta intención
`/voice/status`	GET	Estado del servicio Speech-to-Text

Cuadro 7: Endpoints de comandos de voz

A.3.3. Reconocimiento Facial

Endpoint	Método	Descripción
`/face/emotion/upload`	POST	Analiza emociones en imagen subida
`/face/detect`	POST	Detecta rostros en imagen
`/face/attributes`	POST	Devuelve atributos faciales
`/face/status`	GET	Estado del servicio de reconocimiento facial

Cuadro 8: Endpoints de reconocimiento facial

A.4. Anexo D: Datasets Utilizados

Dataset	Fuente	Descripción
Bitcoin Price	Kaggle	Precios históricos diarios 2012–2021
Body Fat	UCI ML Repo	Mediciones antropométricas de 252 hombres
Avocado Prices	Kaggle	Precios de aguacate en 54 regiones de EE.UU.
Car Prices	Kaggle	Precios de autos usados en India
Wine Quality	UCI ML Repo	Propiedades fisicoquímicas y calidad sensorial
Telco Churn	IBM Sample Data	Datos de clientes en telecomunicaciones
Stroke Prediction	Kaggle	Factores de riesgo de ACV
Hepatitis C	UCI ML Repo	Biomarcadores séricos de hepatitis
Cirrhosis	Mayo Clinic	Estudio de supervivencia de cirrosis biliar

Cuadro 9: Resumen de datasets utilizados

A.5. Anexo E: Comandos de Voz Reconocidos

Comando	Dataset Key
“Jarvis, predice el precio de Bitcoin”	`bitcoin_price`
“Jarvis, calcula el porcentaje de grasa corporal”	`body_fat`
“Jarvis, predice precios de aguacate”	`avocado_prices`
“Jarvis, estima el valor del automóvil”	`car_prices`
“Jarvis, predice la calidad del vino”	`wine_quality`
“Jarvis, analiza el churn de clientes”	`telco_churn`
“Jarvis, predice riesgo de derrame cerebral”	`stroke_risk`
“Jarvis, diagnostica hepatitis C”	`hepatitis_c`
“Jarvis, evalúa estatus de cirrosis”	`cirrhosis_status`

Cuadro 10: Comandos de voz soportados

A.6. Anexo F: Emociones Detectadas

A.6.1. Azure Face API

- **Anger**, **Contempt**, **Disgust**, **Fear**, **Happiness**, **Neutral**, **Sadness**, **Surprise**.

A.6.2. DeepFace

- **Angry**, **Disgust**, **Fear**, **Happy**, **Sad**, **Surprise**, **Neutral**.

A.7. Anexo G: Métricas Consolidadas

Modelo	Tipo	Métrica 1	Métrica 2	Métrica 3	Evaluación
Bitcoin	Regresión	MAPE: 17 %	MAE: 385	RMSE: 699	Bueno
Body Fat	Regresión	R^2: 0.998	MAE: 0.20	RMSE: 0.29	Excelente
Wine Quality	Clasificación	Acc: 54 %	Prec: 52 %	F1: 51 %	Moderado
Telco Churn	Clasificación	Acc: 80 %	Prec: 65 %	F1: 60 %	Bueno
Stroke	Clasificación	Acc: 95 %	Prec: 100 %	Rec: 2 %	Limitado

Cuadro 11: Resumen de métricas principales

A.8. Anexo H: Guía de Pruebas

```python
import unittest
from api.services.model_service import model_registry

class TestModelService(unittest.TestCase):
    def test_load_body_fat_model(self):
        model = model_registry.load_model('body_fat')
        self.assertIsNotNone(model)

    def test_prediction_body_fat(self):
        data = {
            'Age': 23, 'Weight': 154.25,
            'Height': 67.75, 'Neck': 36.2,
            'Chest': 93.1, 'Abdomen': 85.2,
```

```
            'Hip': 94.5, 'Thigh': 59.0,
            'Knee': 37.3, 'Ankle': 21.9,
            'Biceps': 32.0, 'Forearm': 27.4,
            'Wrist': 17.1
        }
        result = model_registry.predict('body_fat', data)
        self.assertIn('prediction', result)
        self.assertGreater(result['prediction'], 0)
        self.assertLess(result['prediction'], 50)

if __name__ == '__main__':
    unittest.main()
```

Listing 3: Ejemplo de test unitario

```
import requests

response = requests.post(
    'http://localhost:8000/ml/predict',
    json={
        'dataset_key': 'body_fat',
        'data': {...}
    }
)
assert response.status_code == 200
assert 'prediction' in response.json()

response = requests.post(
    'http://localhost:8000/voice/parse',
    params={'text': 'Jarvis, predice el precio de Bitcoin'}
)
assert response.status_code == 200
assert response.json()['dataset_key'] == 'bitcoin_price'
```

Listing 4: Test de API completa

A.9. Anexo I: Troubleshooting

```
pip install -r requirements.txt
```

```
netstat -ano | findstr :8000
taskkill /PID <PID> /F
```

```
set GOOGLE_APPLICATION_CREDENTIALS=path/to/credentials.json
```

Problemas de CORS

- Verificar que el frontend esté en `http://localhost:8080`.
- Revisar la configuración CORS en `backend/api/main.py`.
- Reiniciar ambos servidores.

A.10. Anexo J: Código de Ejemplo

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import pickle

df = pd.read_csv('data/raw/my_dataset.csv')
X = df.drop('target', axis=1)
y = df['target']

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

model = RandomForestClassifier(
    n_estimators=100,
    max_depth=10,
    random_state=42
)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))

with open('backend/reports/my_model.pkl', 'wb') as f:
    pickle.dump(model, f)
```

Listing 5: Pipeline de entrenamiento