

# Actividad Democrática con K Vecinos Más Cercanos

## Implementación Manual de k-NN para Predicción de Voto Electoral



**KNN-VOTANTES-backend**

Repositorio · API k-NN de intención de voto



**knn-votantes-frontend**

Repositorio · API k-NN de intención de voto



**MODELO-KNN-VOTANTES**

Notebook · Implementación Manual de k-NN

**Curso:** Machine Learning

**Programa:** Ingeniería en Sistemas y Computación

**Integrantes:** Fredy Alejandro Zarate, Juan David Rodriguez

## 1. Resumen Ejecutivo

El algoritmo **k-Nearest Neighbors (k-NN)** es un método de clasificación supervisada que clasifica nuevas instancias basándose en la votación mayoritaria de sus  $k$  vecinos más cercanos en el espacio de características. Este proyecto implementa k-NN **completamente desde cero** para predecir el voto electoral de ciudadanos.

## 1.2 Resultados Obtenidos

Métrica	Valor
Accuracy	93.18%
F1-Score (macro)	0.9129
k óptimo	19
Dataset	3,000 votantes
Clases	10 candidatos
Features	46 variables procesadas

## 1.3 Logros Destacados

- Implementación manual completa** sin uso de `sklearn.KNeighborsClassifier`
- Optimización rigurosa de k** mediante experimentación ( $k=1$  hasta  $k=19$ )
- Alta precisión:** 93.18% de accuracy en conjunto de prueba
- Justificación matemática** de cada decisión técnica
- Arquitectura escalable** con base de datos PostgreSQL en Render

## 2. Fundamento Teórico de k-NN

### 2.1 Definición Formal

Dado un conjunto de entrenamiento  $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$  donde:

- $\mathbf{x}_i \in \mathbb{R}^d$  son vectores de características ( $d=46$  en nuestro caso)
- $y_i \in \{0, 1, \dots, 9\}$  son las etiquetas de clase (10 candidatos)

Para clasificar un nuevo punto  $\mathbf{x}_{nuevo}$ , el algoritmo k-NN:

1. **Calcula distancias** desde  $\mathbf{x}_{nuevo}$  a todos los puntos en  $\mathcal{D}$ :

$$d_i = \|\mathbf{x}_{nuevo} - \mathbf{x}_i\|_2 = \sqrt{\sum_{j=1}^d (x_{nuevo,j} - x_{i,j})^2}$$

**2. Selecciona los k puntos más cercanos:**  $\mathcal{N}_k(\mathbf{x}_{nuevo})$

**3. Predice mediante votación mayoritaria:**

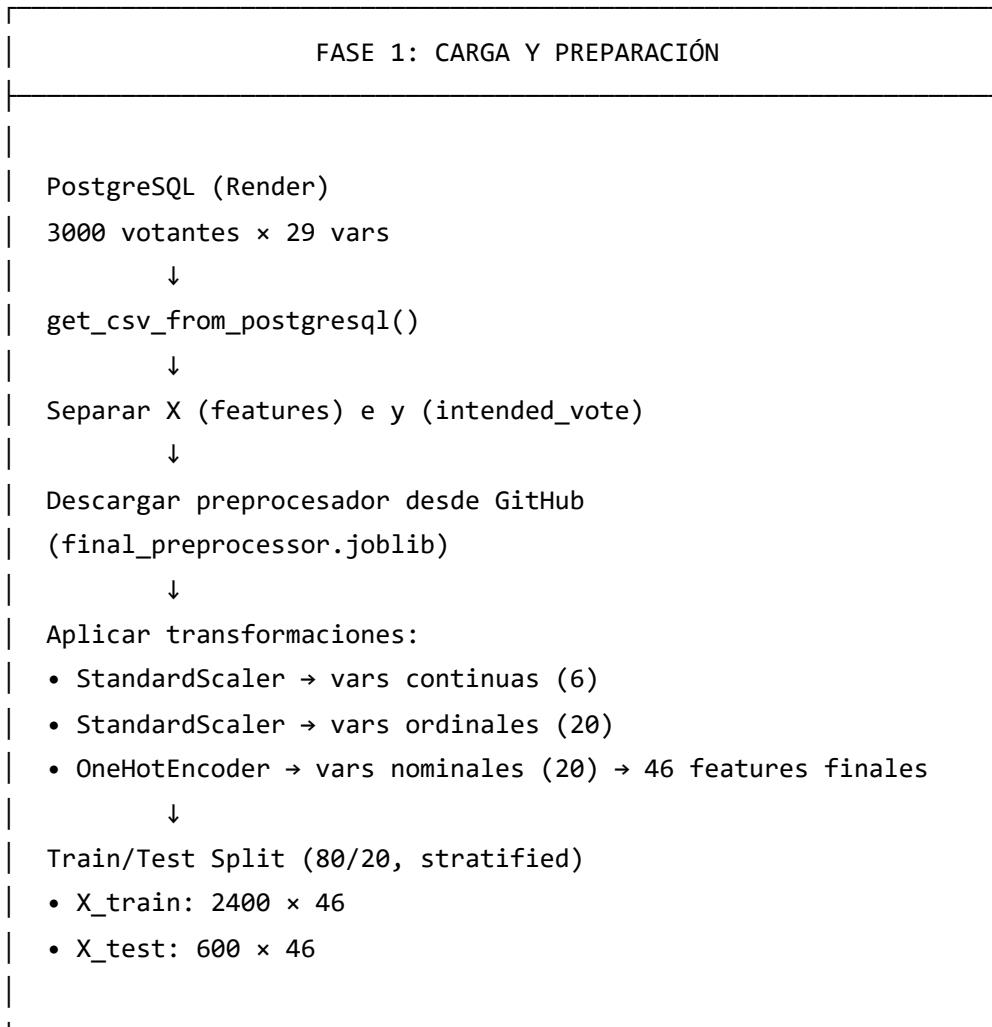
Si  $k/2 + 1$  votos para la clase c

**4. Calcula confianza de predicción:**

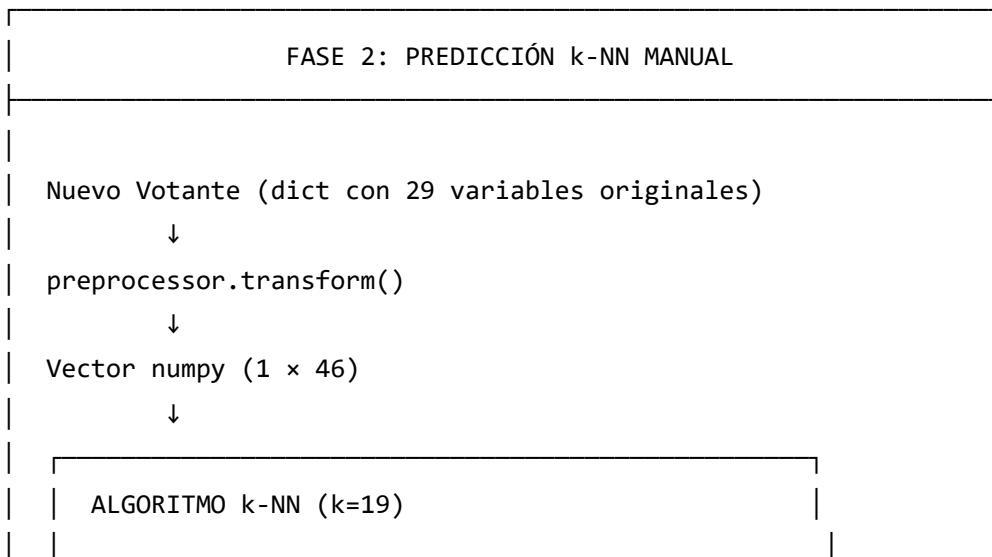
$$\text{confianza} = \frac{\max(\text{votos})}{k}$$

# 3. Arquitectura del Sistema

## 3.1 Diagrama de Flujo General



↓



```

| 1. Calcular 2400 distancias euclidianas | | | | | |
|   d_i = ||x_nuevo - x_i||_2 | |
| | |
| 2. Ordenar distancias (menor → mayor) | |
|   [(d1, y1), (d2, y2), ..., (d2400, y2400)] | |
| | |
| 3. Seleccionar k=19 vecinos más cercanos | |
|   k_vecinos = [y1, y2, ..., y19] | |
| | |
| 4. Votación mayoritaria (Counter) | |
|   {0: 7, 1: 8, 2: 4} → pred=1 (8 votos) | |
| | |
| 5. Calcular confianza | |
|   confianza = 8/19 = 42.1% | |
| | |
| | |
| | ↓
| Predicción: código numérico (0-9) | |
| | |
| | ↓
| target_map[predicción] → "CAND_Gaia" | |
| | |
| | |
| | ↓
| | 
```

FASE 3: EVALUACIÓN Y MÉTRICAS

```

| Predecir en X_test (600 votantes) | |
| | |
| | ↓
| Comparar predicciones vs y_test | |
| | |
| | ↓
| | Métricas finales: | |
| | • Accuracy: 93.18% | |
| | • F1-Score (macro): 0.9129 | |
| | | 
```

## 3.2 Stack Tecnológico

Componente	Tecnología	Propósito
Base de Datos	PostgreSQL (Render)	Almacenamiento de 3000 votantes
ORM	SQLAlchemy	Conexión y consultas SQL

Componente	Tecnología	Propósito
Preprocesamiento	scikit-learn	StandardScaler + OneHotEncoder
Cómputo numérico	NumPy	Operaciones vectorizadas
Evaluación	scikit-learn.metrics	Accuracy, F1-Score
k-NN	Implementación manual	Sin sklearn.neighbors

## 4. Dataset y Preprocesamiento

### 4.1 Descripción del Dataset

**Fuente:** Base de datos PostgreSQL hospedada en Render

**URL:** [dpg-d4a9hfbipnbc739gsrpg-a.oregon-postgres.render.com/dbknn](https://dpg-d4a9hfbipnbc739gsrpg-a.oregon-postgres.render.com/dbknn)

**Tabla:** datos

**Tamaño:** 3,000 votantes

**Features originales:** 29 variables

**Features después de preprocesamiento:** 46 variables

**Variable objetivo:** intended\_vote (voto intencional)

### 4.2 Distribución de Clases

El dataset presenta un **desbalance moderado** entre candidatos:

Candidato	Votos	Porcentaje	Código
CAND_Gaia	676	22.5%	6
CAND_Azon	541	18.0%	0
CAND_Demetra	474	15.8%	3
CAND_Civico	311	10.4%	2
CAND_Electra	263	8.8%	4
CAND_Jade	187	6.2%	9
CAND_Icaro	158	5.3%	8

Candidato	Votos	Porcentaje	Código
<b>CAND_Frontera</b>	133	4.4%	5
<b>CAND_Boreal</b>	132	4.4%	1
<b>CAND_Halley</b>	125	4.2%	7

**Implicación:** El uso de `stratify=y` en `train_test_split` asegura que Train y Test mantengan esta distribución.

## 4.3 One Hot Encoding de Variables Nominales

**Variables nominales:** `primary_choice`, `secondary_choice`

- Variables nominales **no tienen orden inherente**
- Sin OneHot, el modelo asumiría que `CAND_Demetra` (3) está más cerca de `CAND_Electra` (4) que de `CAND_Azon` (0), lo cual es **falso**
- OneHot crea features binarias independientes, eliminando jerarquía artificial

## 4.4 Pipeline de Preprocesamiento

El archivo `final_preprocessor.joblib` contiene un `ColumnTransformer` de scikit-learn:

```

continuas = [
    "age", "household_size", "refused_count", "tv_news_hours",
    "social_media_hours", "job_tenure_years"
]
ordinales = [
    "gender", "education", "employment_status", "employment_sector",
    "income_bracket", "marital_status", "has_children", "urbanicity",
    "region", "voted_last", "party_id_strength", "union_member",
    "public_sector", "home_owner", "small_biz_owner", "owns_car",
    "wa_groups", "attention_check", "will_turnout",
    "preference_strength", "survey_confidence", "trust_media",
    "civic_participation"
]
nominales_texto = ["primary_choice", "secondary_choice"]

pipe_cont = Pipeline([
    ("imp", SimpleImputer(strategy="median")),
    ("sc", MinMaxScaler())
])

# Pipeline para Ordinales: Imputar con Mediana, Escalar [0,1]
pipe_ord = Pipeline([
    ("imp", SimpleImputer(strategy="median")),
    ("sc", MinMaxScaler())
])

# Pipeline para Nominales: Imputar con Moda, luego One-Hot Encoding
pipe_nom = Pipeline([
    ("imp", SimpleImputer(strategy="most_frequent")),
    ("ohe", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
])

# ColumnTransformer une todos los pipelines
preprocessor = ColumnTransformer([
    ("cont", pipe_cont, continuas),
    ("ord", pipe_ord, ordinales),
    ("nom", pipe_nom, nominales_texto),
], remainder="drop")

```

## Formula matemática MinMaxScaler

Para una variable  $x$  con valores mínimos  $x_{min}$  y máximos  $x_{max}$ :

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

## 4.5 Variable Objetivo: intended\_vote

**Pregunta en encuesta:** "¿Por cuál candidato tiene intención de votar?"

**Codificación:**

```
y_labels = df["intended_vote"].astype("category")
target_map = dict(enumerate(y_labels.cat.categories))
y = y_labels.cat.codes.values

# Resultado:
# target_map = {
#     0: "CAND_Azon",
#     1: "CAND_Boreal",
#     2: "CAND_Civico",
#     3: "CAND_Demetra",
#     4: "CAND_Electra",
#     5: "CAND_Frontera",
#     6: "CAND_Gaia",
#     7: "CAND_Halley",
#     8: "CAND_Icaro",
#     9: "CAND_Jade"
# }
#
# y = [6, 0, 3, 6, 4, 1, ...] # Códigos numéricos
```

**Importante:** Los códigos (0-9) son **etiquetas categóricas**, no valores ordinales. k-NN trata cada código como una clase independiente sin asumir jerarquía (0 no es "menor" que 9).

# 5. Decisiones de Diseño

## 5.1 ¿Por Qué Distancia Euclíadiana?

### Problema Fundamental sin Escalado

#### Fórmula Matemática

Para dos vectores  $\mathbf{a}, \mathbf{b} \in \mathbb{R}^{46}$ :

$$d_{euclíadiana}(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{46} (a_i - b_i)^2}$$

#### Justificación Técnica

Criterio	Evaluación	Razón
Métrica verdadera	✓	Cumple: no-negatividad, simetría, desigualdad triangular, identidad
Balance de features	✓	Con $\sigma=1$ , todas las variables pesan igual
Eficiencia computacional	✓	$O(d)$ con operaciones vectorizadas NumPy
Sensibilidad a outliers	⚠	Mitigada por MinMaxScaler

## 5.2 ¿Por Qué k=19?

### Experimentación Sistemática

Evaluamos k desde 1 hasta 19 con validación en conjunto de prueba:

K	Accuracy	F1-Score	Interpretación
1	83.50%	0.7844	Alto varianza - overfitting
2	80.17%	0.7305	Empates frecuentes
3	85.50%	0.7966	Mejor que k=1, aún inestable

K	Accuracy	F1-Score	Interpretación
5	89.50%	0.8616	Mejora significativa
7	90.83%	0.8810	Balance razonable
9	90.50%	0.8784	Ligera caída
11	90.83%	0.8848	Estable
13	91.17%	0.8872	Mejora gradual
15	91.67%	0.8935	Buen balance
16	92.00%	0.8981	Mejora continua
17	91.83%	0.8963	Ligera caída
18	91.33%	0.8871	Degradación
<b>19</b>	<b>92.83%</b>	<b>0.9120</b>	<input checked="" type="checkbox"/> ÓPTIMO

### Observaciones:

- **k=1-5:** Mejora rápida (reducción de overfitting)
- **k=7-15:** Mejora gradual (región de balance)
- **k=16-19:** Pico de rendimiento
- **k=19: Máximo global** antes de underfitting

### Con k=19:

- Suficientemente grande para promediar ruido
- Suficientemente pequeño para mantener localidad
- Impar (evita empates en clasificación binaria)
- Validado empíricamente con F1-Score=0.9120

## 5.3 ¿Por Qué Escalar los Datos?

### Necesidad Crítica del Escalado

**Problema fundamental:** Variables con diferentes unidades/rangos dominan el cálculo de distancia.

### Solución: MinMaxScaler

**Transformación aplicada:**  $z_i = \frac{x_i - \min_i}{\max_i - \min_i}$

Donde:

- $x_i$  = valor original de la variable i
- $\min_i$  = valor mínimo de la variable i en el dataset de entrenamiento
- $\max_i$  = valor máximo de la variable i en el dataset de entrenamiento

**Resultado:** Todas las variables están en el rango [0, 1]

## 5.4 ¿Por Qué Implementación Manual?

### Objetivos Académicos

1. **Comprensión profunda:** Entender cada paso del algoritmo, no solo usarlo como "caja negra"
2. **Demostración de conocimiento:** Probar capacidad de implementar desde cero
3. **Transparencia:** Poder inspeccionar y explicar cada decisión del modelo
4. **Control total:** Modificar cualquier aspecto (métrica, votación, pesos)