IA para detectar las amenazas que los firewalls ignoran

Hack & Beers Gijón 2026

Dilema del defensor

Defensores

Tienen que acertar
siempre

Un error = compromiso

Atacantes

Solo necesitan
un error

Tiempo ilimitado

La IA invierte la ecuación

01 Tiempo de detección

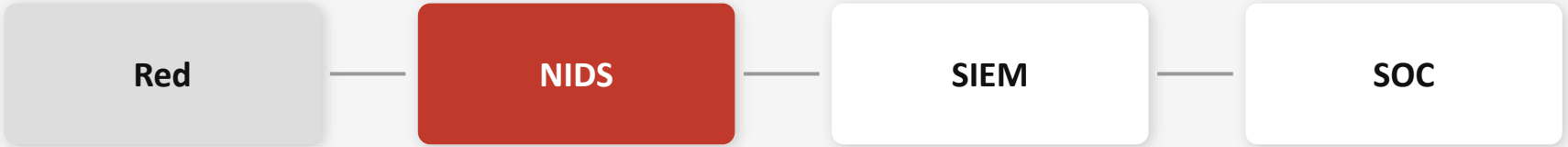
02 Número de alertas

03 Zero-day attacks

Proteger, Empoderar, Avanzar: Cómo la IA puede invertir el dilema del defensor (2024 - Google)
https://services.google.com/fh/files/misc/how-ai-can-reverse-defenders-dilemma.pdf

¿Qué es un NIDS?

Network Intrusion Detection System



Segunda línea de defensa. Detecta lo que el firewall dejó pasar.

NIDS (red) ≠ HIDS (host)

Dos paradigmas de detección

Firmas

+ Rápido y preciso

- Ciego a zero-day

- Modificación Manual

Anomalías

- Lento y Falsos positivos

+ Detecta zero-day

+ Aprendizaje automático

Complementarios: Firmas → Anomalías

Vamos a ello …

media.arrospide.dev/hnb2026

Carga de datos (kaggle)

Name	Type	Description
49 unique values	integer 41% Float 20% Other (19) 39%	49 unique values
srcip	nominal	Source IP address
sport	integer	Source port number
dstip	nominal	Destination IP address
dsport	integer	Destination port number
proto	nominal	Transaction protocol
state	nominal	Indicates to the state and its dependent protocol, e.g. ACC, CLO, CON, ECO, ECR, FIN, INT, MAS, PAR, ...

HACKATHON

The Pokémon Company - PTCG AI Battle Challenge...

Analyze data and agentic play supportin...

$240,000 3 months to go

Kaggle | The World's AI Proving Ground
kaggle.com

Carga de datos (UNSW_NB15)

```
import kagglehub
path = kagglehub.dataset_download("mrwellsdavid/unsw-nb15")
```

Kaggle | UNSW_NB15 by the Australian Centre for Cyber Security (ACCS)
kaggle.com/datasets/mrwellsdavid/unsw-nb15

Carga de datos

```
import pandas as pd
df = pd.read_csv(path + "/UNSW_NB15_training-set.csv")
```

```
TARGET = 'label'

X = df_clean.drop(columns=[TARGET])
y = df_clean[TARGET]
```

pandas | Python Data Analysis Library
pandas.pydata.org

Modelos supervisados

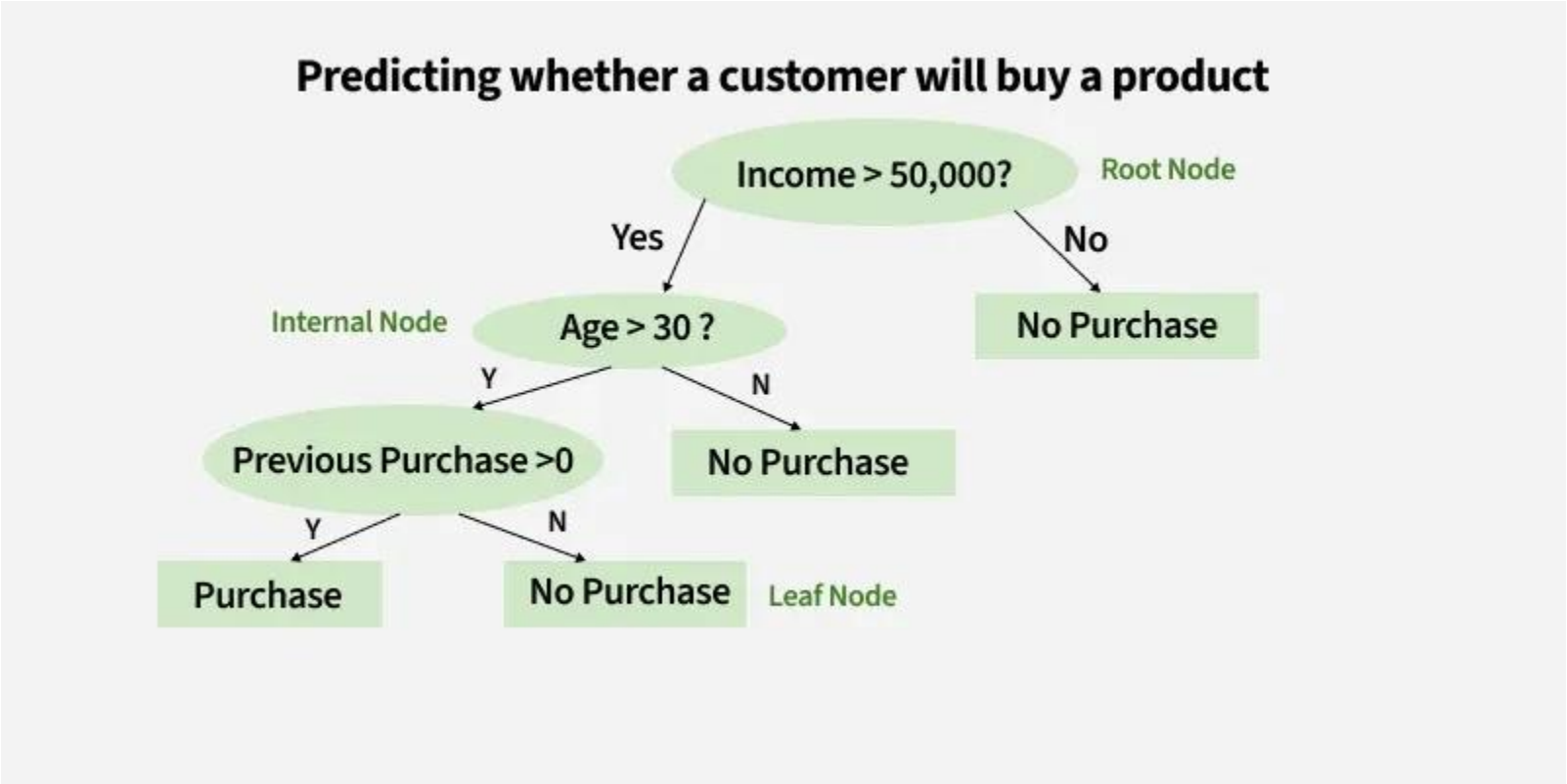
```
from sklearn import …

model = …
model.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

Preprocessing

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()  # Otros como MinMaxScaler
X_scaled = scaler.fit_transform(X)
```

Balance de clases (y) / NaNs / Variables categoricas (X)

Decision Tree (concepto)

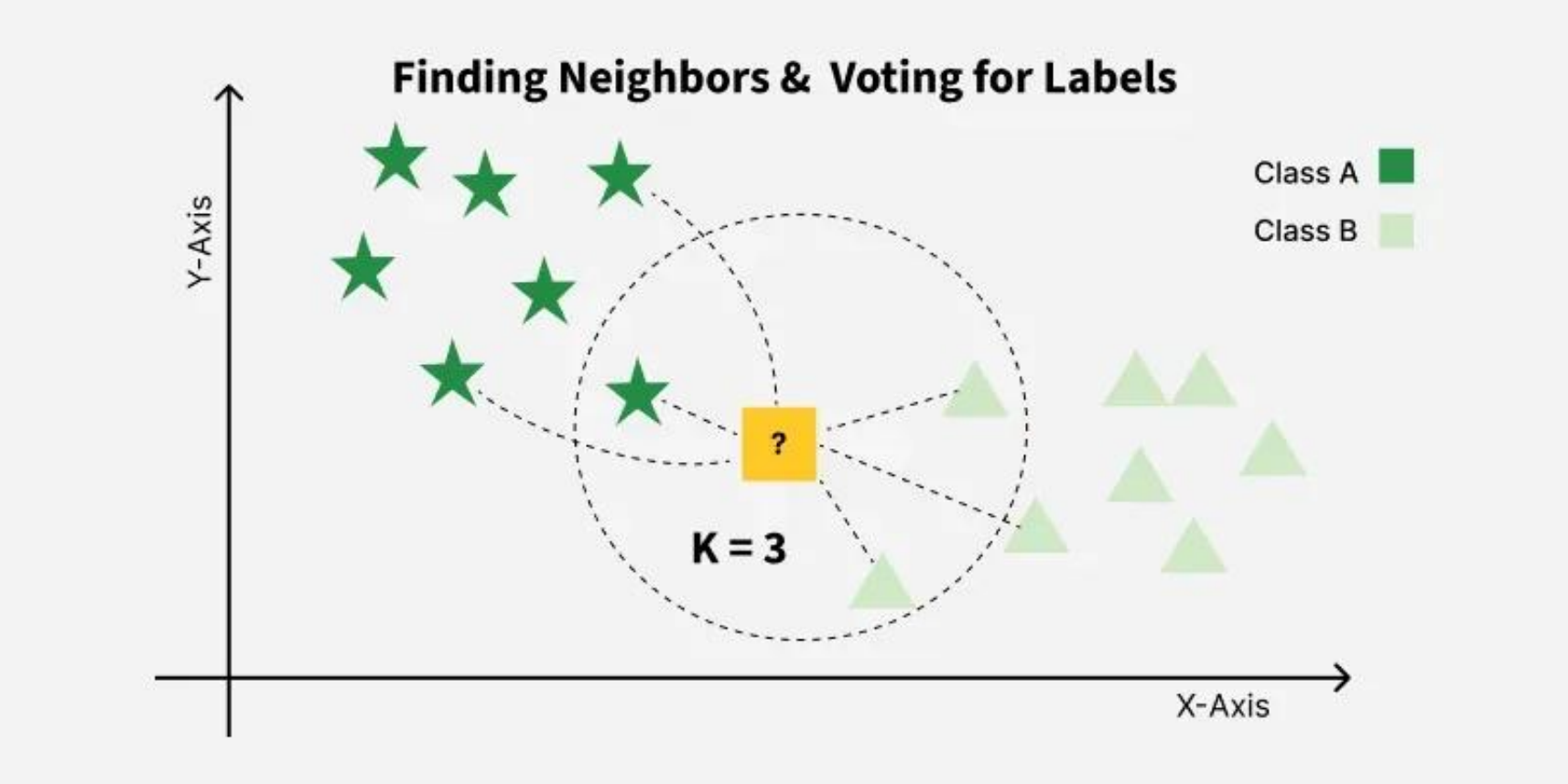


Random Forest (Código)

```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(n_estimators=100)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

F1 = 0.962 Recall = 0.960 Precision = 0.964

K-Nearest Neighbors (concepto)

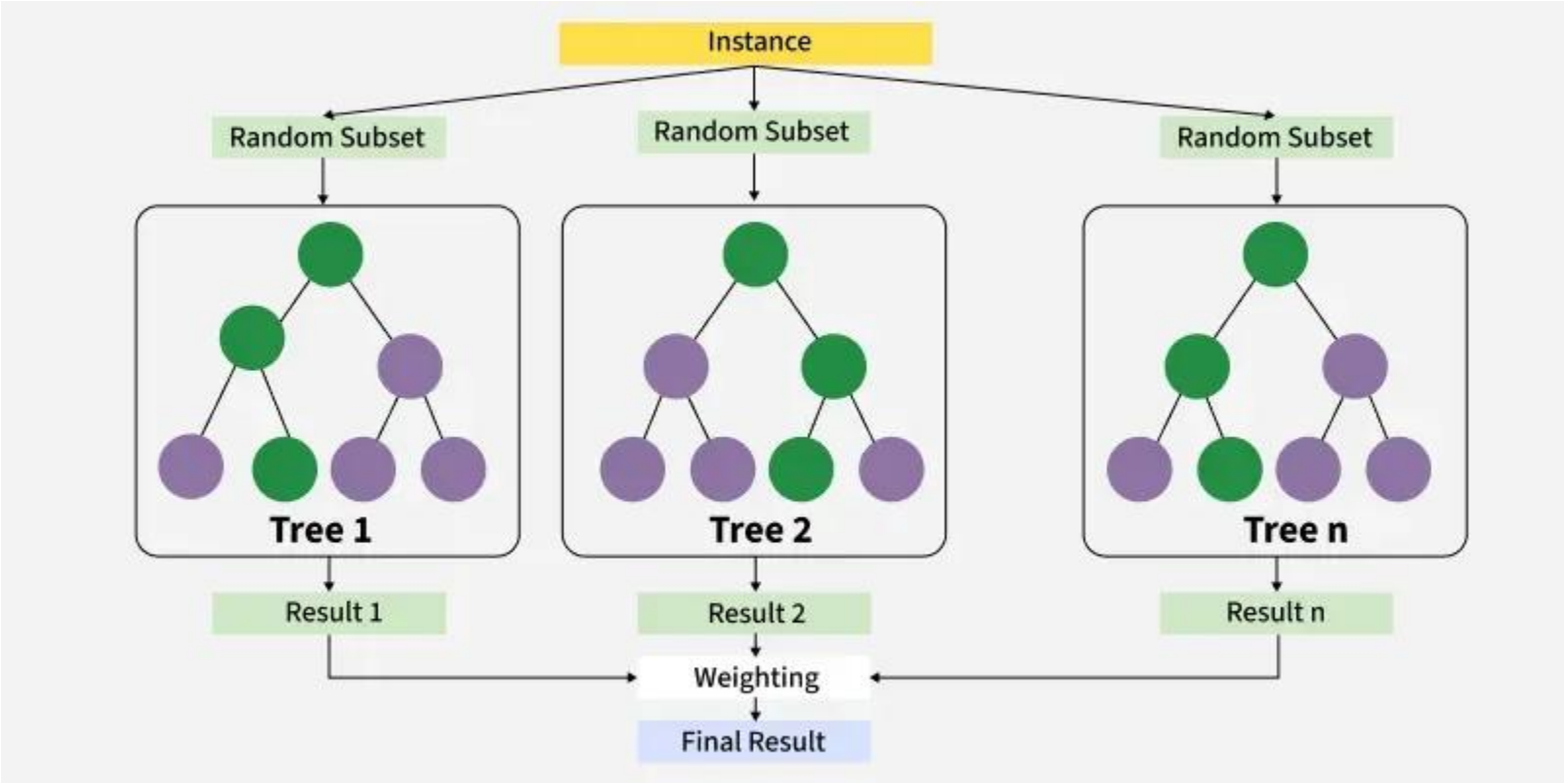


K-Nearest Neighbors (Código)

```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=3)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

F1 = 0.937 Recall = 0.934 Precision = 0.939

XGBoost (concepto)

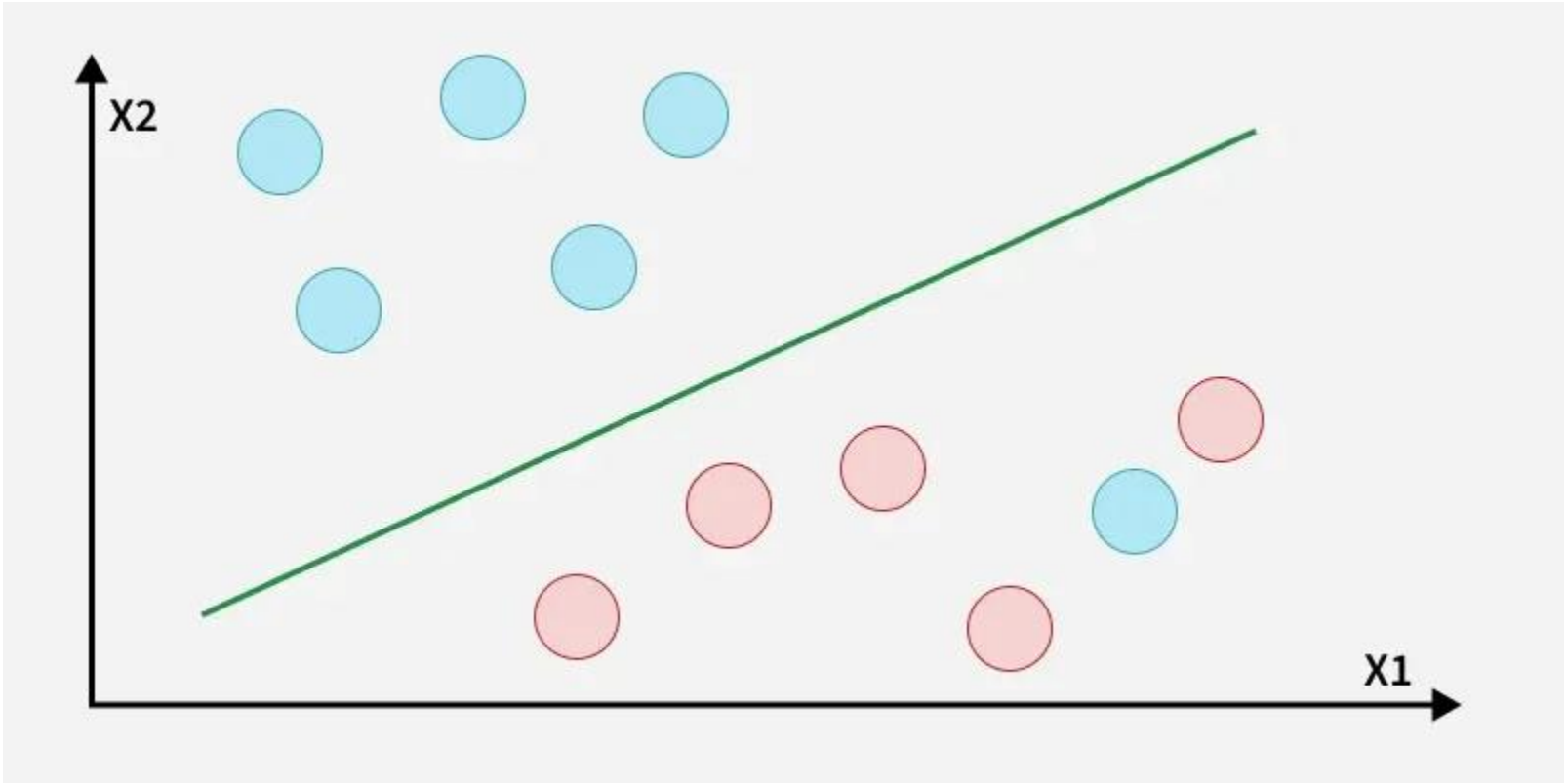


XGBoost (Código)

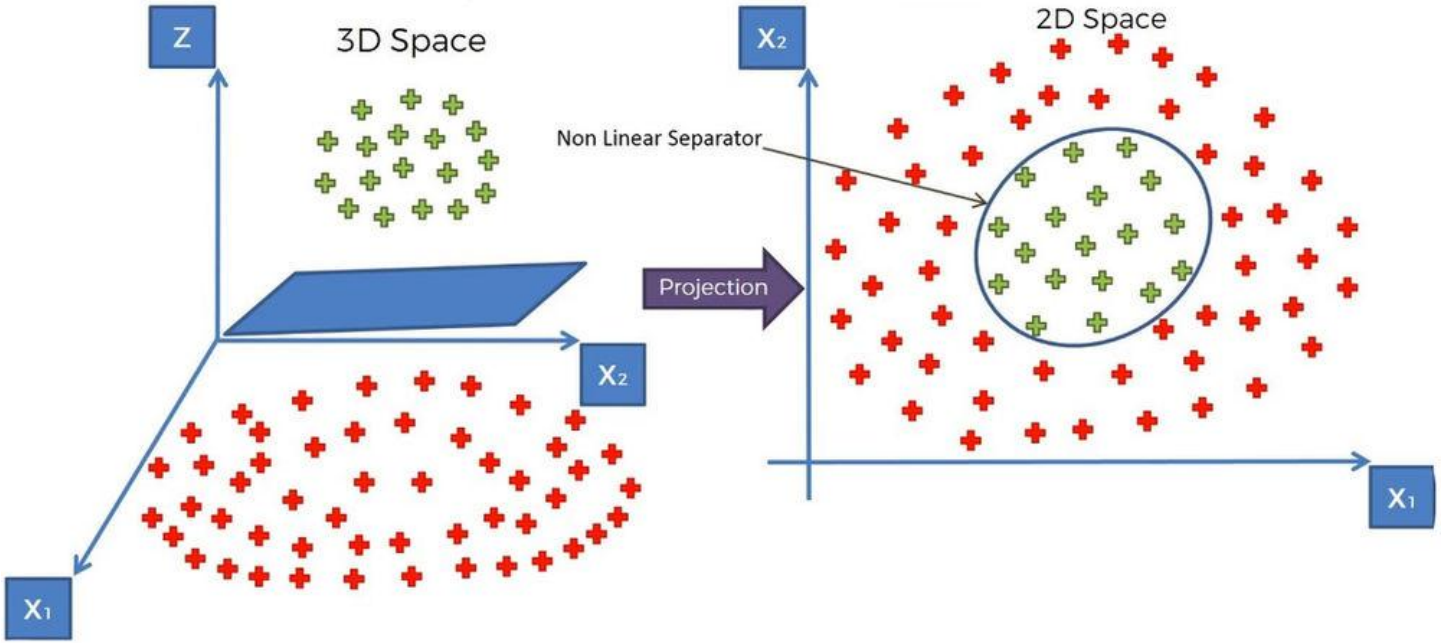
```
import xgboost as xgb

clf = xgb.XGBClassifier(n_estimators=300)

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

F1 = 0.953 Recall = 0.944 Precision = 0.958

Support Vector Machines (concepto)



Support Vector Machines (Kernels)

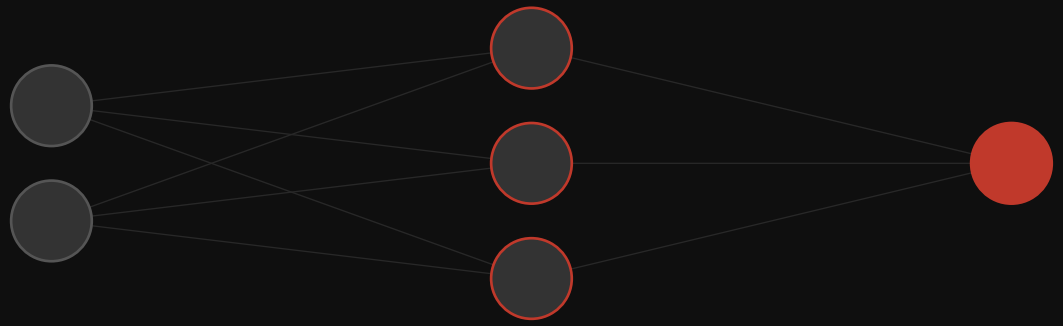


Support Vector Machines (Código)

```
from sklearn.svm import SVC

clf = SVC(kernel='linear')

clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

F1 = 0.953 Recall = 0.944 Precision = 0.958

Red neuronal (concepto)

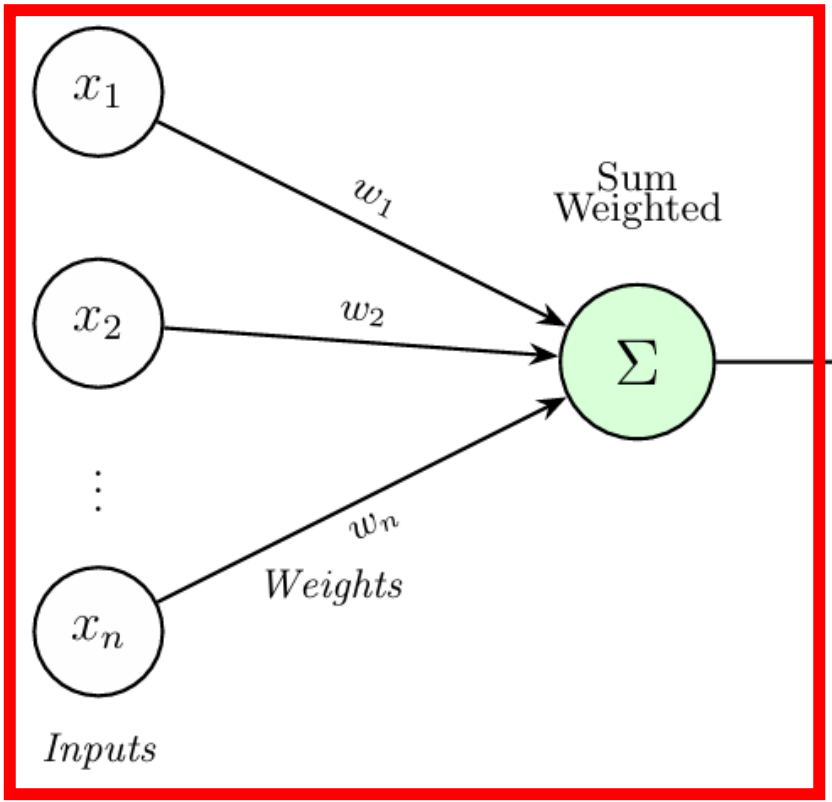


Input
(49 features)

Capa oculta

Output
(normal / ataque)

Neuronas



Name	Graph	Expression
relu		$f(x) = \max(0, x)$
tanh		$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
leaky_relu		$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$
elu		$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$

Regresión Lineal

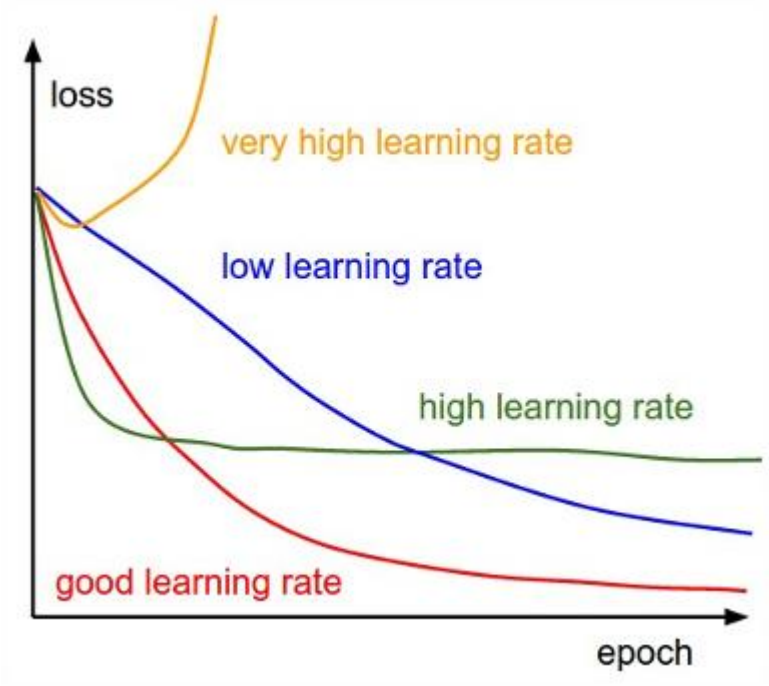
Red Neuronal (Código)

```
import torch.nn as nn # Facebook

class SimpleNNClassifier(nn.Module):
    def __init__(self, input_size):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(input_size, 16),
            nn.ReLU(), # ...
            nn.Linear(16, 1),
        )
    def forward(self, x):
            return self.layers(x)
```

Red Neuronal (Entrenar)

```
import torch.optim as optim

nn_model = SimpleNNClassifier(X_train.shape[1])
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.Adam(nn_model.parameters(), lr=0.001)

for epoch in range(20):
    optimizer.zero_grad()
    y_pred = nn_model(X_train)
    loss = criterion(y_pred, y_train)
    loss.backward()
    optimizer.step()
```

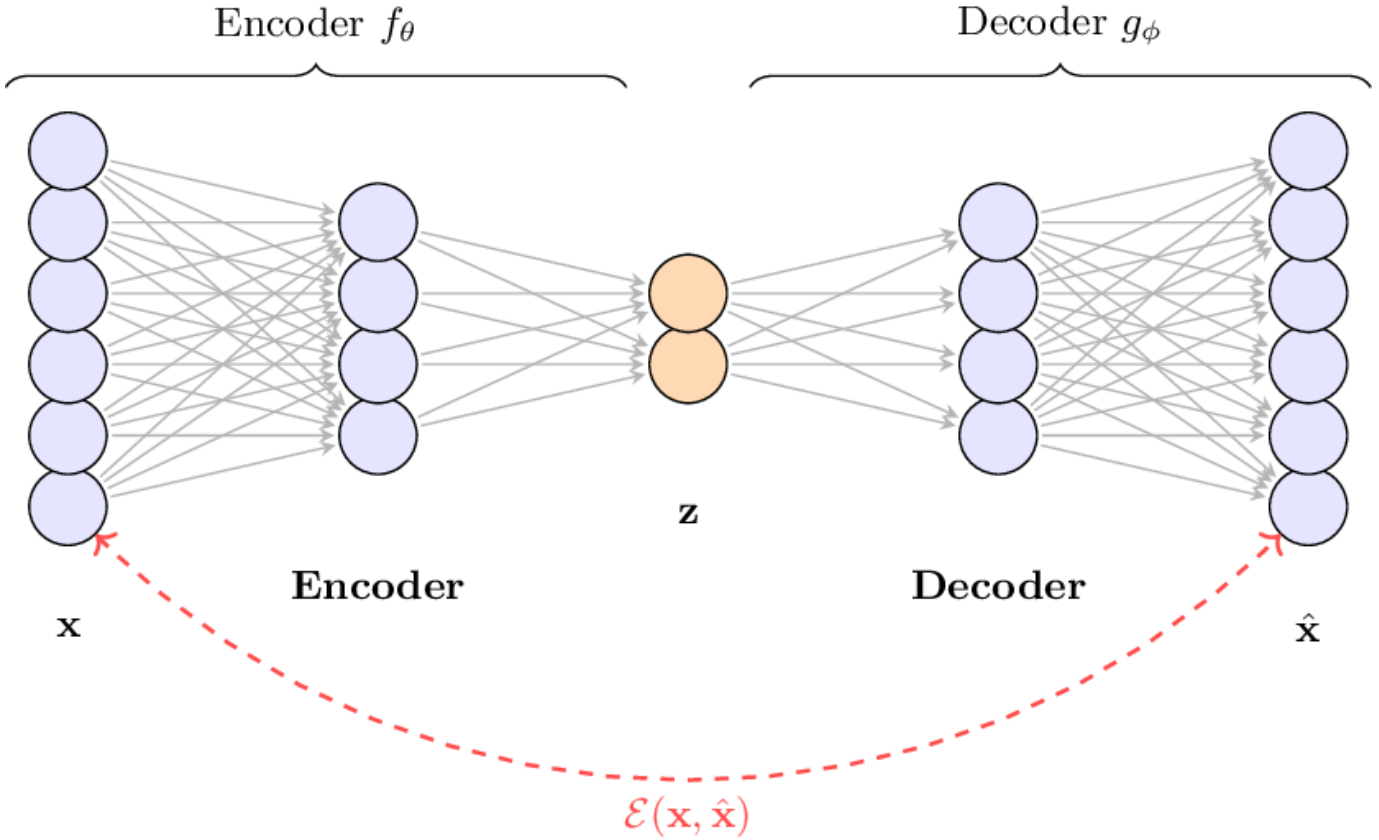
Learning rate y resultados



Resultados

```
=== Red Neuronal Simple ===
  Accuracy : 0.9245
  Precision: 0.9227
  Recall   : 0.9625
  F1       : 0.9421
```

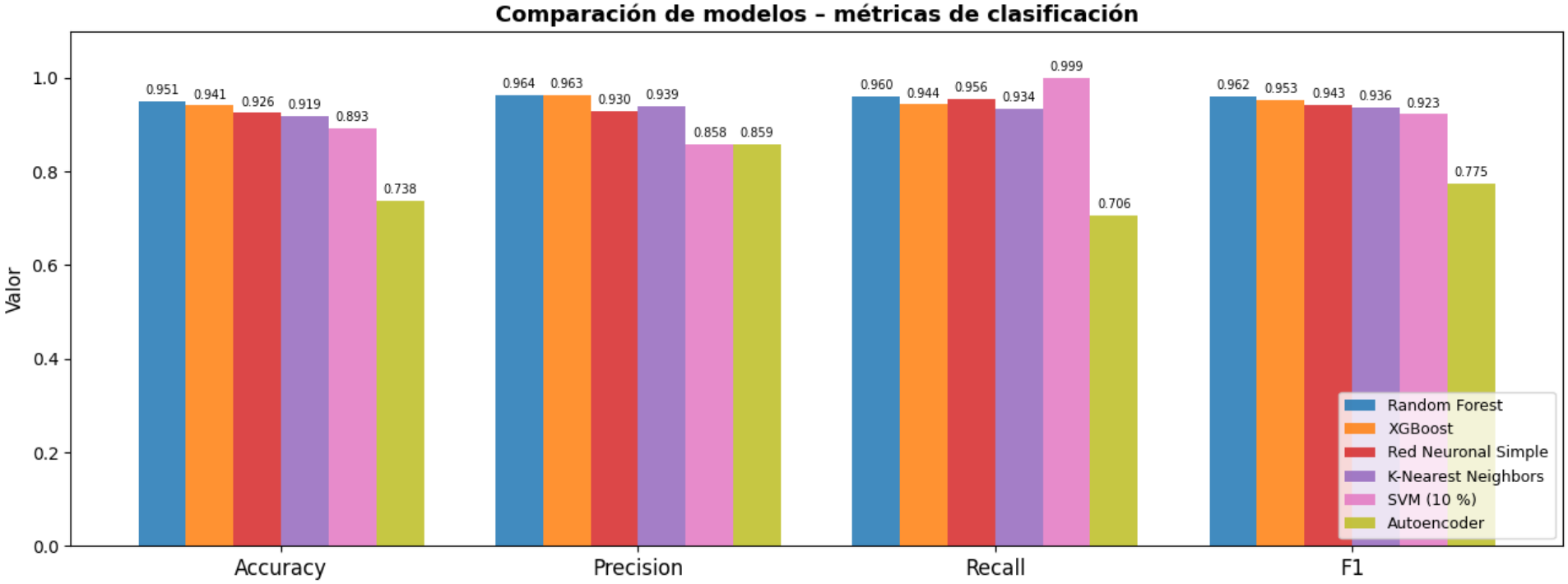
Autoencoder



Pruébalo — ¿qué puedes cambiar?

`epochs = 20  vs  100`	*¿Converge igual de rápido?*
`N° Neuronas y capas`	*¿Mejor o peor detección?*
`Otra activación`	*¿Cambia el error?*

Comparación de modelos



Tendencias

- Adversarial attacks
- XAI
- LLMs en SOC
- IA Agentica

La IA no reemplaza al firewall
ni al analista.

Los potencia.