



**INSTITUTO TECNOLÓGICO DE CULIACÁN  
INGENIERÍA EN SISTEMAS COMPUTACIONALES**



**ASIGNATURA:**

INTELIGENCIA ARTIFICIAL

**DOCENTE:**

ZURIEL DATHAN MORA FELIX

**DOCUMENTACION**

**TAREA:**

DETECTOR DE SPAM

**ALUMNOS:**

FLORES BELTRAN JESUS GUADALUPE

JOSÉ MANUEL URÍAS ESCOBAR

**FECHA DE ENTREGA:**

12/10/2025

## DETECTOR DE SPAM Y NO SPAM

En este documento estaré recolectando información, código, e investigaciones que me sirvieron para desarrollar el proyecto utilizando el entrenamiento de un modelo de IA para que detecte mediante patrones, si un correo es spam o no.

Bueno, primeramente estuve investigando que el lenguaje que me ofrece las mejores ventajas es Python, porque?, porque es el que mas librerías tiene a la hora de trabajar con cosas relacionadas a IA.

Asi que bien, mediante esa investigación, descubrí que existen variedad de librerías, las que mas me llamaron la atención fueron las siguientes:

- Pandas: librería que me permite manipular una basta cantidad de datos mediante código.
- Numpy
- Unicodedata
- TfIdVectorizer
- Tensorflow: la librería que hace la magia de las IA, desarrollada por Google, es una librería basta para trabajar con redes neuronales y entrenamiento de IAs.
- re

```
pip install keras-preprocessing
```

Bien, entonces, lo que hice fue buscar un dataset con información de mensajes que son SPAM y mensajes que no son SPAM, es de extensión .csv, y lo primero que hice fue leer ese archivo de la siguiente forma:

```
import pandas as pd

# Esta es una función que sirve para leer un archivo CSV en un DataFrame
df = pd.read_csv("Datasets/emails/correos_es_prueba.csv")

# muestro sus primeras filas

print("Las primeras filas del DataFrame son:")

print(df.columns)

print(df.head())

# si quiero extraer una columna específica del dataset

print(df[["Email No.", "Prediction"]].head())
```

## CONFIGURACION DE ENTORNO VIRTUAL EN PYTHON

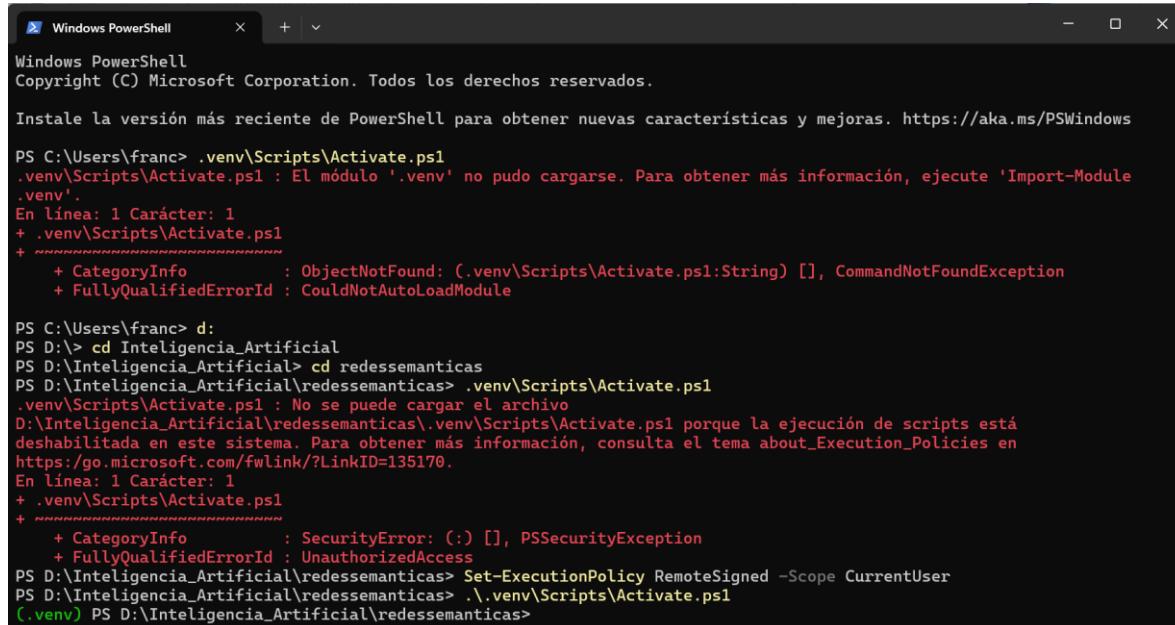
Para trabajar en este proyecto, lo que hicimos fue crear un entorno virtual de Python para no terminar dañando nuestras compatibilidades de librerías ni nada, así trabajamos de forma experimental y segura.

Los comandos para crear el entorno son los siguientes:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

```
python -m venv .venv
```

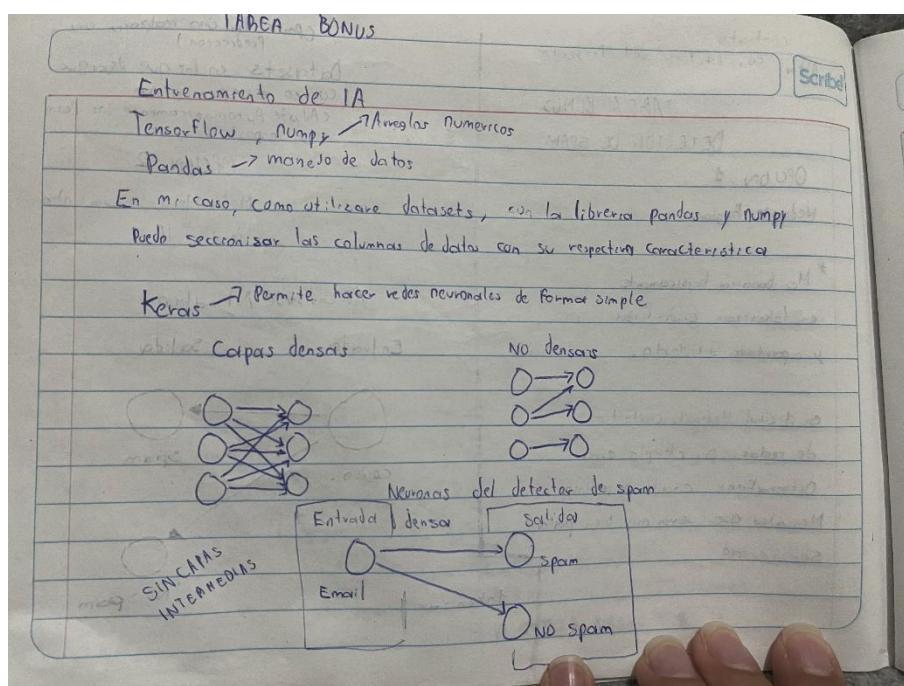
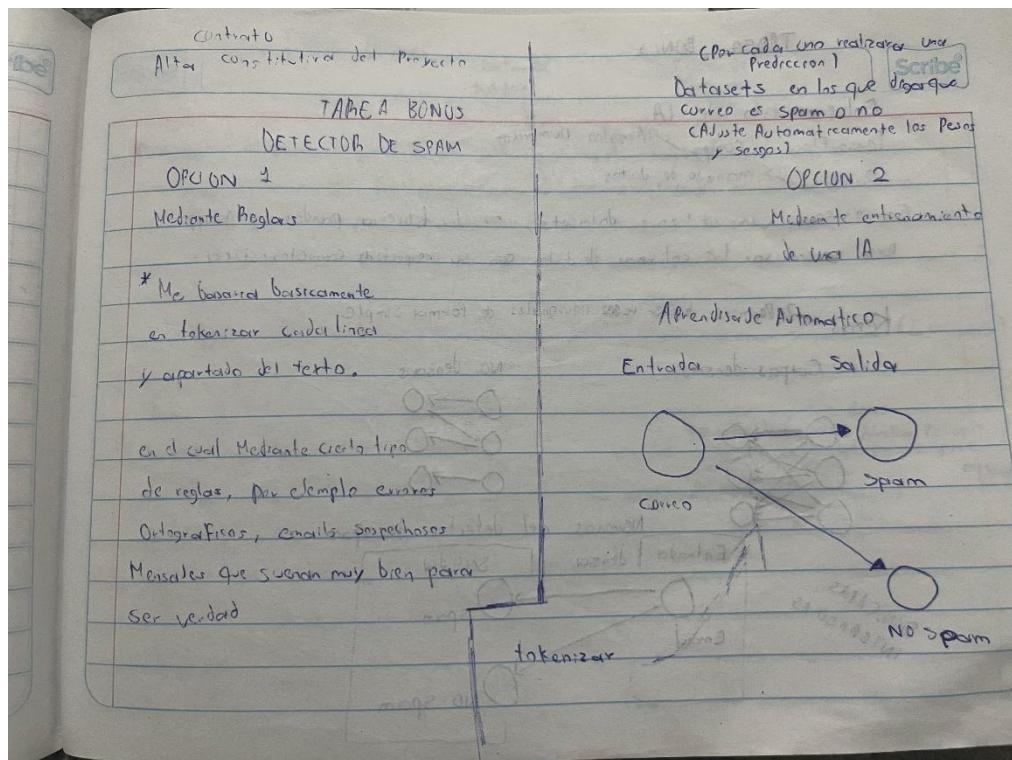
```
.venv\Scripts\Activate.ps1
```



The screenshot shows a Windows PowerShell window titled "Windows PowerShell". The command "Set-ExecutionPolicy RemoteSigned -Scope CurrentUser" was run, resulting in the message "Set-ExecutionPolicy : El módulo '.venv' no pudo cargarse. Para obtener más información, ejecute 'Import-Module .venv'. En linea: 1 Carácter: 1 + .venv\Scripts\Activate.ps1 +----- + CategoryInfo : ObjectNotFound: (.venv\Scripts\Activate.ps1:String) [], CommandNotFoundException + FullyQualifiedErrorId : CouldNotAutoLoadModule". The command "python -m venv .venv" was run, followed by ".venv\Scripts\Activate.ps1". The final prompt shows the virtual environment has been activated, indicated by the "(.venv)" prefix in the PowerShell window title bar.

Después de preparar el entorno, entender mas o menos como leer los csv con pandas, lo que nos toca es lo siguiente, conocer como funciona una red neuronal, para ello realice una documentación en la libreta la cual es la siguiente

Para entender el funcionamiento de las redes neuronales, los sistemas inteligentes, hice estos apuntes de el trabajo de cada capa.



## TAREA BONUS

Capas de  
Salidas

Capas de  
Entradas

Scribe

# Capa = tf.keras.layers.Dense(Units=2, input\_shape=[1])

Modelo Secuencial: Proceso que toma en cuenta el orden de los ~~pasos~~ datos o decisiones.

- Decisión Actual depende de la secuencia de estados pasados anteriores.

modelo = tf.keras.Sequential([Capa]) → modelo = keras.Sequential([Capa])

Bien, Ahora al modelo se le debe decir como queremos que procese los

Matemáticas

Permite saber como Ajustar Pesos y  
seguir, para que poco a poco

modelo.compile(optimizer=tf.keras.optimizer.Adam(0.1), loss='mean\_squared\_error')

tarza de  
Aprendizaje

3

Entrenamiento

historial = modelo.fit

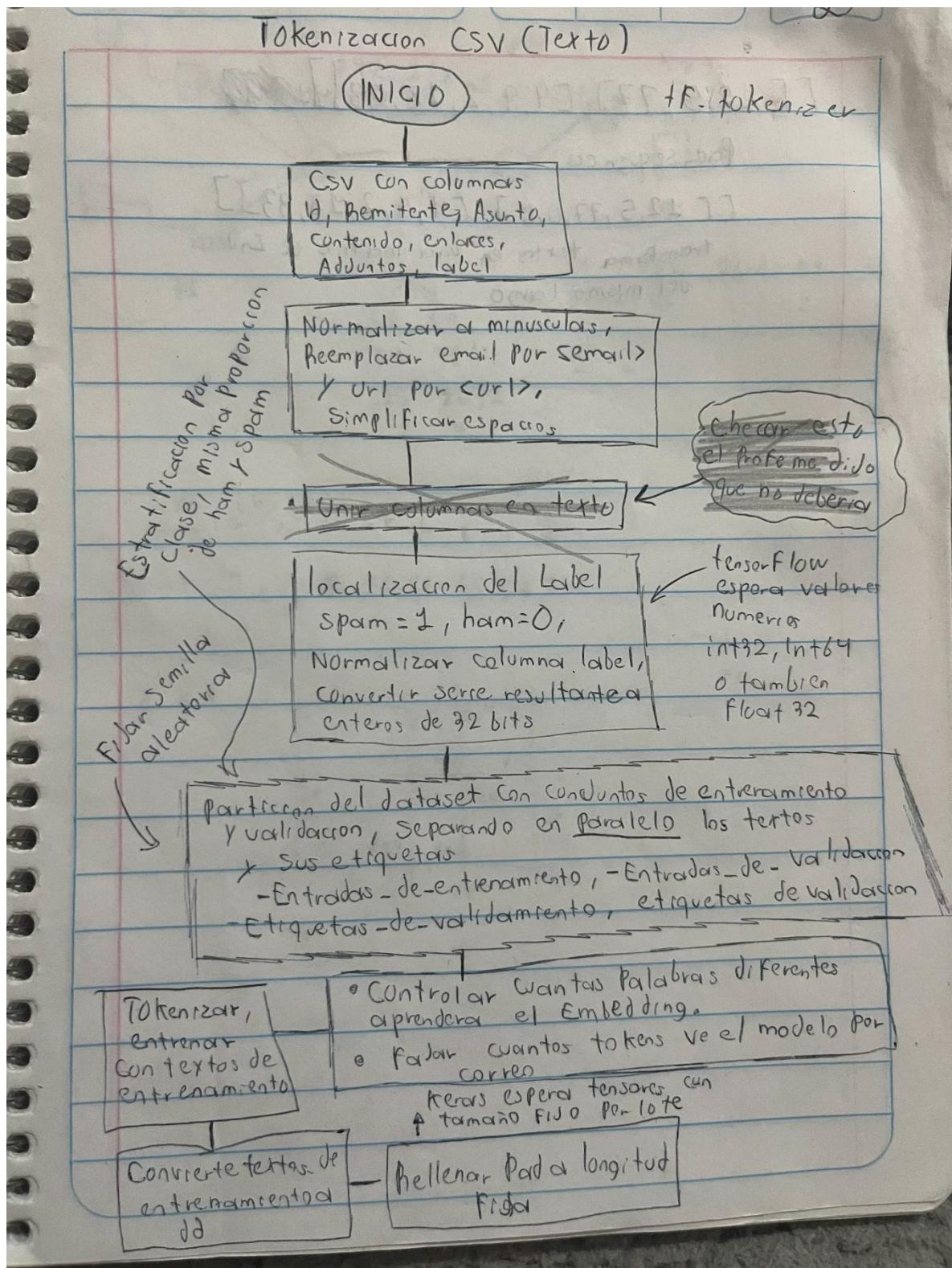
Apuntes de como se programa las capas en tensorflow, con sus capas de entrada y salida, todo esto para entender como usar tensorflow. La creación del modelo, y el porque del uso del modelo secuencial, y modelo\_compile, donde definimos como este agente aprenderá.

Y a continuación un pequeño ejemplo de lectura de csv con pandas en Python.

### C-LASE LECTURA DE CSV

```
import pandas as pd
CSV_PATH = "Dataset/emails_spam.csv"
df = pd.read_csv(CSV_PATH, encoding="utf-8")
print("Filas, columnas:", df.shape)
print("N columnas:", len(df.columns))
print(df.head())
```

Ahora, para ingresar datos a tensorflow necesitamos tokenizar los datos, para lo cual diseña el siguiente diagrama de flujo



Ahora toca definir el modelo, para ello lo hicimos en la clase ModeloSpam del archivo Red\_Neuronal

Una clase que define, entrena, evalúa y usa un **modelo binario de spam/ham** en Keras/TensorFlow para texto tokenizado.

Constructor:

```
ModeloSpam(  
    tam_vocabulario=5000,    tamaño del vocabulario (coincide con Tokenizer.num_words)  
    max_longitud=200,        longitud fija de cada secuencia (pad_sequences)  
    dim_embedding=64,        dimensión del vector por palabra  
    unidades=64,            neuronas en la capa densa intermedia  
    dropout=0.5,            regularización contra overfitting  
    lr=0.001                learning rate del optimizador Adam  
)
```

Arquitectura:

### Arquitectura

Secuencial, pensada para datasets pequeños/medianos:

1. **Input:** shape=(max\_longitud,), dtype=int32 (índices de tokens).
2. **Embedding:** Embedding(tam\_vocabulario, dim\_embedding, mask\_zero=True)
  - mask\_zero=True hace que el modelo **ignore el padding (0)**.
3. **GlobalAveragePooling1D:** promedia los embeddings por posición → vector único por correo.
4. **Dense(ReLU):** unidades neuronas para aprender combinaciones no lineales.
5. **Dropout:** apaga aleatoriamente neuronas en entrenamiento (mejor generalización).
6. **Dense(sigmoid): 1 salida en [0,1] = p(spam).**

Compilacion:

```
optimizer = Adam(lr)  
loss    = "binary_crossentropy"    por tener 1 salida sigmoide  
metrics = ["accuracy", AUC(name="auc")]
```

**Ahora Explico un poco el funcionamiento de los métodos.**

**entrenar(x\_entrenamiento, y\_entrenamiento, x\_validacion, y\_validacion, epocas=30, tam\_lote=8, usar\_callbacks=True)**

- Entrena el modelo con labels **0/1** (no one-hot).
- Callbacks (si usar\_callbacks=True):
  - **EarlyStopping** (monitor val\_auc, patience=4, restaura mejores pesos).
  - **ReduceLROnPlateau** (baja LR si val\_loss no mejora).
- Devuelve el History de Keras.

**evaluar(x\_validacion, y\_validacion)**

- Retorna (loss, accuracy, auc) sobre validación.

**predecir\_desde\_indices(entradas\_padded)**

- Recibe secuencias **ya padded** (n, max\_longitud).
- Devuelve:
  - etiquetas: lista de "spam"/"ham" (umbral 0.5),
  - confianzas: prob. de la clase elegida,
  - p\_spam: probabilidad cruda de spam.

**predecir\_desde\_textos(textos, tokenizador)**

- Tokeniza y hace **padding** internamente con el **mismo** Tokenizer usado en entrenamiento.
- Devuelve lo mismo que predecir\_desde\_indices.

### **Entradas y salidas esperadas**

- **Entradas de entrenamiento/validación:**
  - X: np.ndarray de enteros (n, max\_longitud) (resultado de pad\_sequences).
  - y: np.ndarray de enteros (n,) con valores **0 (ham) / 1 (spam)**.
- **Salida de predicción:**
  - p\_spam en [0,1], etiqueta final por umbral 0.5 y confianza asociada.