



Universidad de Oriente

Núcleo Nueva Esparta

Escuela de Ingeniería y Ciencias Aplicadas

Departamento de Informática

Introducción a la Inteligencia Artificial – 2304574

Calculadora Algebraica

Docente:

Lic. Samuel Rojas

Integrantes:

Br. José Silva. CI: 30230054

Br. Isaac Hernández. CI: 30563299

Br. Emanuel Aponte. CI: 29789773

Guatamare, 02/2025

Documentación del Proyecto

¿Qué es una expresión algebraica?: Son términos matemáticos que contienen números y letras. En combinación con los símbolos de las operaciones matemáticas para obtener fórmulas o ecuaciones, a partir de descripciones hechas mediante palabras. A su vez esas letras pueden ser sumadas, restadas, multiplicadas o divididas por otros números, los cuales pueden ser explícitos o representados también por letras.

Clasificación de las expresiones algebraicas: se establece según su número de términos. En general, se dividen en dos: monomios y polinomios.

- **Monomios:** Las expresiones algebraicas llamadas monomios son aquellas que están compuestas por un solo término. Ejemplo: $2x^2$ $2x^2y^3z$.
- **Polinomios:** Los polinomios son una clasificación de expresiones algebraicas que según la cantidad de términos por la que está formada cambia su nombre: binomio, trinomio, etc.... Estas expresiones algebraicas en general se componen por dos o más términos, es decir, por más de un monomio:
 - **Binomios:** se componen por dos términos separados por un operador matemático. Ejemplo: $a^4b^5 + 3a^2b^2c^7$
 - **Trinomios:** Cuando se denomina trinomio, es una expresión algebraica compuesta por tres monomios: $ab^3 + 5a^2b^7m - 35abx^5$

¿Qué hace el proyecto?

Este proyecto es un programa que utiliza inteligencia artificial para analizar expresiones algebraicas (como $3x + 2y$ o $x^2 - 4x - 4$) y clasificarlas en dos categorías:

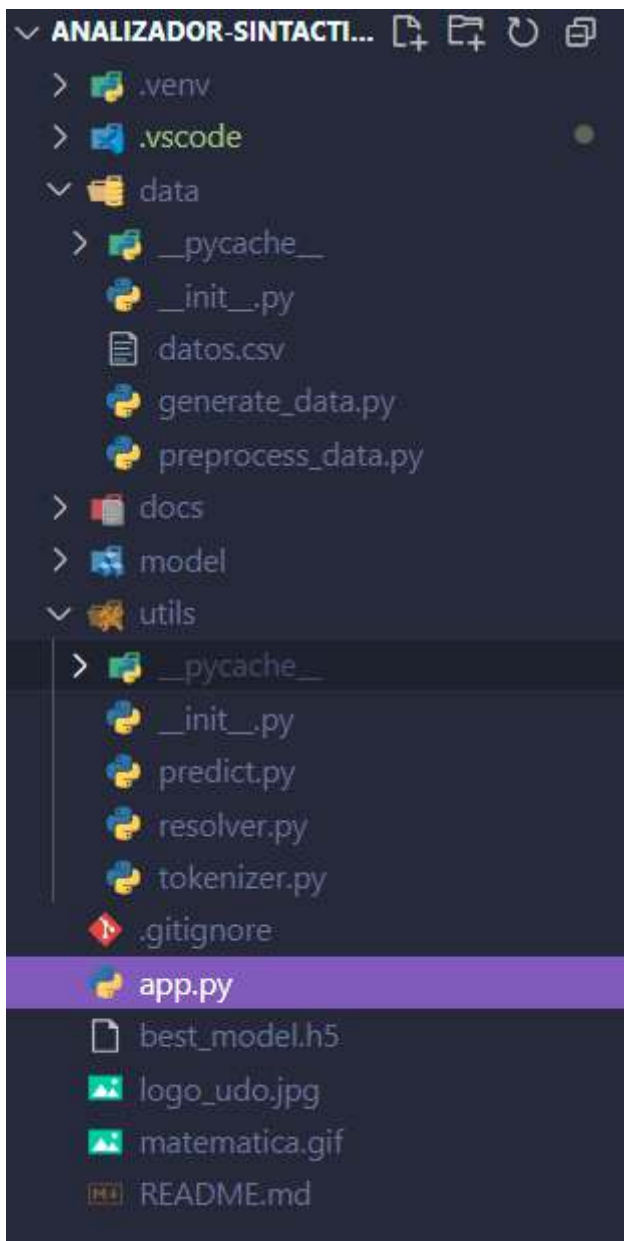
1. **Tipo de expresión:** Por ejemplo, si es un monomio, binomio, trinomio, polinomio y si es una ecuación.
2. **Operación principal:** Si es una suma, resta, multiplicación o división.

Además, también dependiendo del tipo de expresión, realiza los distintos métodos para reducir la operación (factorización, expansión, igualación). El programa “aprende” a hacer esto a partir de un conjunto de datos que contiene ejemplos de expresiones algebraicas y sus clasificaciones

Estructura de carpeta

```
Analizador_sintacticoEA/      # Carpeta raíz del proyecto
|
|— data/                      # Carpeta para almacenar datos
|  |— __init__.py
|  |— datos.csv              # Archivo con expresiones y etiquetas
|  |— generate_data.py       # Script para cargar datos desde el CSV
|  |— preprocess_data.py     # Script para preprocesar los datos
|— docs/                     # Manual de usuario
|— model/                    # Carpeta para el modelo de red neuronal
|  |— __init__.py
|  |— build_model.py         # Script para construir el modelo
|  |— train_model.py        # Script para entrenar el modelo
|
|— utils/                    # Carpeta para utilidades adicionales
|  |— __init__.py
|  |— resolver.py           # script para resolver las expresiones
|  |— predict.py            # Script para hacer predicciones
```

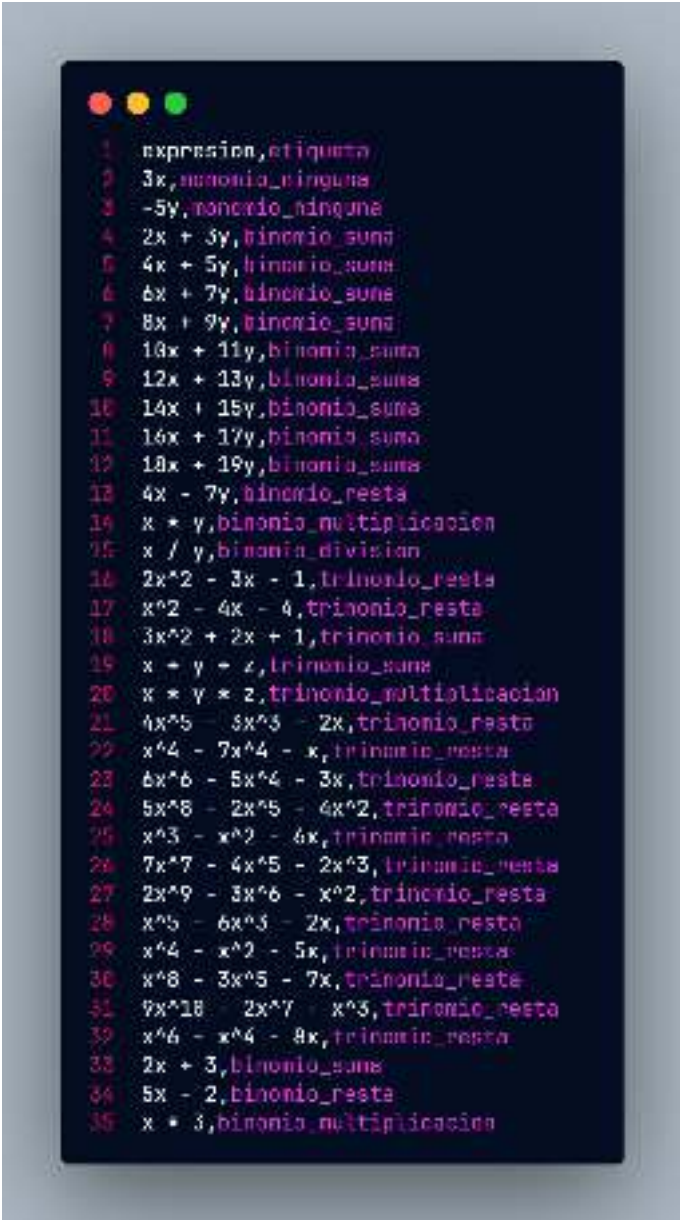
- | └─ tokenizer.py # Script para tokenización y normalización
- | └─ .gitignore # Archivo de git
- | └─ app.py # Archivo principal que ejecuta el programa
- | └─ best_model.h5 # guarda el modelo entrenado en cada llamada
- | └─ matemática.gif
- | └─ logo_udo.jpg # imagen del logo para la presentación
- | └─ README.md # Documentación del proyecto (opcional)



¿Cómo Funciona?

➤ **data/datos.csv:** Este archivo contiene los datos que el programa debe usar para aprender. Es como un libro de ejemplos que tiene 2 columnas:

- **expresión:** una expresión algebraica (por ejemplo, $3x$, $2x + 3y$).
- **etiqueta:** la clasificación de la expresión (monomio_ninguna, binomio_suma)



```
1  expresion,etiqueta
2  3x,monomio_ninguna
3  -5y,monomio_ninguna
4  2x + 3y,binomio_suma
5  4x + 5y,binomio_suma
6  6x + 7y,binomio_suma
7  8x + 9y,binomio_suma
8  10x + 11y,binomio_suma
9  12x + 13y,binomio_suma
10 14x + 15y,binomio_suma
11 16x + 17y,binomio_suma
12 18x + 19y,binomio_suma
13 4x - 7y,binomio_resta
14 x * y,binomio_multiplicacion
15 x / y,binomio_division
16 2x^2 - 3x - 1,trinomio_resta
17 x^2 - 4x - 4,trinomio_resta
18 3x^2 + 2x + 1,trinomio_suma
19 x + y - z,trinomio_suma
20 x * y * z,trinomio_multiplicacion
21 4x^5 - 3x^3 - 2x,trinomio_resta
22 x^4 - 7x^4 - x,trinomio_resta
23 6x^6 - 5x^4 - 3x,trinomio_resta
24 5x^8 - 2x^5 - 4x^2,trinomio_resta
25 x^3 - x^2 - 4x,trinomio_resta
26 7x^7 - 4x^5 - 2x^3,trinomio_resta
27 2x^9 - 3x^6 - x^2,trinomio_resta
28 x^5 - 6x^3 - 2x,trinomio_resta
29 x^4 - x^2 - 5x,trinomio_resta
30 x^8 - 3x^5 - 7x,trinomio_resta
31 9x^18 - 2x^7 - x^3,trinomio_resta
32 x^6 - x^4 - 8x,trinomio_resta
33 2x + 3,binomio_suma
34 5x - 2,binomio_resta
35 x * 3,binomio_multiplicacion
```

➤ data/generate_data.py: contiene una función para cargar los datos desde el archivo CSV.

- Función load_data_from_csv(file_path)

❖ **Propósito:** Carga los datos de entrenamiento desde un archivo CSV.

❖ **Entrada:** la ruta al archivo CSV (file_path)

❖ **Salida:**

- **Expresiones:** una lista de expresiones algebraicas
- **Etiquetas:** una lista de etiquetas correspondientes a las expresiones

❖ **¿Cómo Funciona?**

1. Usa pandas para leer el archivo CSV.
2. Extrae las columnas “*expresión*” y “*etiqueta*” como listas.
3. Devuelve las listas de expresiones y etiquetas

```

1 import pandas as pd
2
3 def load_data_from_csv(file_path):
4     """
5     Carga los datos de entrenamiento desde un archivo CSV.
6
7     Args:
8         file_path (str): Ruta al archivo CSV.
9
10    Returns:
11        expresiones (list): Lista de expresiones algebraicas.
12        etiquetas (list): Lista de etiquetas correspondientes.
13    """
14    # Leer el archivo CSV
15    df = pd.read_csv(file_path)
16
17    # Obtener las columnas como listas
18    expresiones = df['expresion'].tolist()
19    etiquetas = df['etiqueta'].tolist()
20
21    return expresiones, etiquetas

```

➤ **data/preprocess_data.py:** este archivo se encarga de preprocesar los datos antes de entrenar el modelo. Preprocesar hace referencia a que procesa sus datos de entrada para producir una salida que se utiliza como entrada en otro programa, es decir, convierte los datos brutos en un formato que el computador pueda procesar.

- ❖ **Tokenización:** convierte las expresiones en números (por ejemplo, 3x se convierte en [3, x])
- ❖ **Padding:** asegura que todas las expresiones tengan la misma longitud añadiendo ceros al final si es necesario

- ❖ **Conversión de etiquetas:** convierte las etiquetas (como *monomio_ninguna*) en números y luego en un formato especial llamado “*one-hot-encoding*”, que es más fácil de procesar para el programa

```
1 from tensorflow.keras.preprocessing.text import Tokenizer
2 from tensorflow.keras.preprocessing.sequence import pad_sequences
3 from tensorflow.keras.utils import to_categorical
4 import numpy as np
5
6 def preprocess_data(expresiones, etiquetas):
7     # Tokenización de las expresiones
8     tokenizer = Tokenizer(char_level=True)
9     tokenizer.fit_on_texts(expresiones)
10    sequences = tokenizer.texts_to_sequences(expresiones)
11
12    # Padding para que todas las secuencias tengan la misma longitud
13    max_len = max(len(seq) for seq in sequences)
14    X = pad_sequences(sequences, maxlen=max_len, padding='max')
15
16    # Convertir etiquetas a números
17    tipos = sorted(set(etiquetas.split('_')[0] for etiqueta in etiquetas)))
18    operaciones = sorted(set(etiquetas.split('_')[1] if '_' in etiqueta else 'ninguna' for etiqueta in etiquetas))
19
20    tipo_to_index = {tipo: idx for idx, tipo in enumerate(tipos)}
21    operacion_to_index = {operacion: idx for idx, operacion in enumerate(operaciones)}
22
23    y_tipos = np.array([tipo_to_index(etiqueta.split('_')[0]) for etiqueta in etiquetas])
24    y_operaciones = np.array([operacion_to_index(etiqueta.split('_')[1] if '_' in etiqueta else 'ninguna') for etiqueta in etiquetas])
25
26    # Convertir etiquetas a one-hot encoding
27    y_tipos = to_categorical(y_tipos, num_classes=len(tipo_to_index))
28    y_operaciones = to_categorical(y_operaciones, num_classes=len(operacion_to_index))
29
30    return X, y_tipos, y_operaciones, tokenizer, max_len, tipo_to_index, operacion_to_index
```

- **model/build_model.py:** este construye el “*cerebro*” del programa, es decir, aquí se construye la red neuronal.

- ❖ **Capa de entrada:** recibe las expresiones convertidas en números
- ❖ **Embedding:** convierte esos números en vectores (lista de números) que representan mejor la información
- ❖ **Capas convolucionales y LSTM:** estas capas ayudan a extraer características importantes de las expresiones
- ❖ **Capas densas:** procesan la información para tomar decisiones

❖ **Salidas:** El programa predice dos cosas:

- El tipo de expresión (por ejemplo, monomio, binomio).
- La operación principal (por ejemplo, suma, resta)

```
1 from tensorflow.keras.models import Model
2 from tensorflow.keras.layers import Input, Embedding, Conv3D, MaxPooling3D, LSTM, Dense, Dropout, GlobalMaxPooling3D, concatenate
3
4 def build_model(vocab_size, embedding_dim, num_types, num_operations, max_len):
5     # Entrada
6     input_layer = Input(shape=(max_len,))
7
8     # Paso de embedding
9     embedding = Embedding(input_dim=vocab_size, output_dim=embedding_dim)(input_layer)
10
11     # capa convolucional
12     conv = Conv3D(128, 3, activation='relu')(embedding)
13     pool = MaxPooling3D()(conv)
14
15     # Capa LSTM
16     lstm = LSTM(128, return_sequences=True)(pool)
17     lstm = Dropout(0.5)(lstm)
18
19     # Pooling global
20     global_pool = GlobalMaxPooling3D()(lstm)
21
22     # Capa densa
23     dense = Dense(64, activation='relu')(global_pool)
24     dense = Dropout(0.5)(dense)
25
26     # Salida para el tipo de expresión
27     output_types = Dense(num_types, activation='softmax', name='types')(dense)
28
29     # Salida para la operación
30     output_operations = Dense(num_operations, activation='softmax', name='operations')(dense)
31
32     # Modelo
33     model = Model([input_layer], outputs=[output_types, output_operations])
34
35     # Compilar el modelo
36     model.compile(
37         optimizer='adam',
38         loss={'types': 'categorical_crossentropy', 'operations': 'categorical_crossentropy'},
39         metrics={'types': 'accuracy', 'operations': 'accuracy'})
40
41     return model
```

- **model/train_model.py:** Este archivo se encarga de entrenar el modelo con los datos.

- ❖ **Entrenamiento:** usa los datos preprocesados para enseñarle al modelo cómo clasificar las expresiones
- ❖ **Callback:** durante el entrenamiento, el programa verifica si está mejorando y guarda el mejor modelo

```
1 from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
2
3 def train_model(model, X, y, epochs=100, batch_size=32, validation_split=0.2):
4     # Callbacks
5     early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
6     model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', save_best_only=True)
7
8     # Entrenamiento
9     history = model.fit(
10         X, y,
11         epochs=epochs,
12         batch_size=batch_size,
13         validation_split=validation_split,
14         callbacks=[early_stopping, model_checkpoint])
15
16     return history
```

- **utils/predict.py:** Este archivo permite usar el modelo entrenado para hacer predicciones

- ❖ **Tokenización:** Convierte una nueva expresión en números
- ❖ **Predicción:** usa el modelo para predecir el tipo y la operación de la expresión
- ❖ **Resultado:** muestra la clasificación (por ejemplo, “la expresión $3x + 2y$ ” es un binomio con operación de suma)

```
1 import numpy as np
2 from tensorflow.keras.preprocessing.sequence import pad_sequences
3
4 def predecir_expresion(model, tokenizer, max_len, tipo_to_index, operacion_to_index, expresion):
5     # Tokenizar la expresión
6     sequence = tokenizer.texts_to_sequences([expresion])
7
8     # Añadir padding a la secuencia
9     padded_sequence = pad_sequences(sequence, maxlen=max_len, padding='post')
10
11     # Hacer la predicción
12     predicciones_tipos, predicciones_operaciones = model.predict(padded_sequence)
13
14     # Obtener la clase predicha para el tipo de expresión
15     indice_tipo = np.argmax(predicciones_tipos)
16     tipo = (list(tipo_to_index.keys()))[(list(tipo_to_index.values()).index(indice_tipo))]
17
18     # Obtener la clase predicha para la operación
19     indice_operacion = np.argmax(predicciones_operaciones)
20     operacion = (list(operacion_to_index.keys()))[(list(operacion_to_index.values()).index(indice_operacion))]
21
22     # Devolver el resultado
23     return tipo, operacion
```

- **utils/tokenizer.py:** contiene funciones adicionales para procesar las expresiones.

- ❖ **Tokenización personalizada:** convierte las expresiones en números de una manera específica.
- ❖ **Normalización:** Limpia las expresiones (por ejemplo, elimina espacios y convierte **2x** en **2*x**).



```
1 from tensorflow.keras.preprocessing.text import Tokenizer
2 import re
3
4 def custom_tokenizer(expresiones):
5     tokenizer = Tokenizer(char_level=False)
6     tokenizer.fit_on_texts(expresiones)
7     return tokenizer
8
9 def normalize_expression(expression):
10     # Normaliza la expresion (ejemplo basico)
11     expression = re.sub(r'\s+', '', expression) # Elimina espacios
12     expression = re.sub(r'(\d)([a-zA-Z])', r'\1*\2', expression) # Convierte 2x a 2*x
13     return expression
```

utils/resolver.py: Este archivo permite usar el modelo entrenado para hacer predicciones

- **Corregir multiplicación implícita:** función para convertir una multiplicación que se debe interpretar. Ejemplo: si se ingresa 3x la función separa la variable del operador (3x -> 3*x).
- **Expresión sin “/left /right”:** la librería sympy (usada para resolver las expresiones) escribe dichas palabras dentro de la expresión, así que se usa esta función para omitir las palabras y tener una salida más entendible.
- **Resolver expresión:** utilizada para resolver la expresión y decir los pasos que se debe realizar

```

1 import sympy as sp
2 import re
3
4 def corregir_multiplicacion_implicita(expresion):
5     """Corrige la notación matemática implícita, reemplazando casos como '3x' con '3*x'."""
6     expresion = re.sub(r'(\d)([a-zA-Z])', r'\1*\2', expresion)
7     expresion = re.sub(r'([a-zA-Z])([a-zA-Z])', r'\1*\2', expresion)
8     expresion = expresion.replace('^', '**')
9     return expresion
10
11 def latex_sin_left_right(expresion):
12     """Convierte una expresión de SymPy a latex sin \left y \right."""
13     return sp.latex(expresion, mode='plain').replace(r'\left', '').replace(r'\right', '')
14
15 def resolver_expresion(expresion):
16     pasos = []
17
18     try:
19         sp.init_printing(use_unicode=True)
20         expresion = corregir_multiplicacion_implicita(expresion)
21
22         if '*' in expresion:
23             izquierda, derecha = expresion.split('*')
24             izquierdo = sp.simplify(izquierda)
25             derecha = sp.simplify(derecha)
26             x = sp.Symbol('x') # Definir variable principal
27             ecuacion = sp.Eq(izquierdo, derecha)
28             pasos.append(f"""Función dada: {sp.latex_sin_left_right(ecuacion)}""")
29
30             # Paso 1: mover todo al lado izquierdo
31             ecuacion_transformada = sp.simplify(izquierdo - derecha)

```

➤ **app.py**: archivo principal que ejecuta todo el programa

- ❖ **Carga los datos**: carga las expresiones y etiquetas
- ❖ **Preprocesa los datos**: prepara los datos en el formato correcto
- ❖ **Construye el modelo**: construye la red neuronal
- ❖ **Entrena el modelo**: usa “*train_model.py*” para enseñarle el modelo cómo clasificar
- ❖ **Hace predicciones**: usa “*predict.py*” para clasificar una nueva expresión ingresada por el usuario.
- ❖ **interfaz gráfica**: utiliza la librería “tkinter” para crear la interfaz, a su vez tenemos la función “animar_gif()” para usar un archivo .gif en la interfaz.

```
1 from data.generate_data import load_data_from_csv
2 from data.preprocess_data import preprocess_data
3 from model.build_model import build_model
4 from model.train_model import train_model
5 from utils.predict import predecir_expresion
6
7 def main():
8     # Cargar datos desde el archivo CSV
9     file_path = "data/entree.csv" # Ruta al archivo CSV
10    expresiones, etiquetas = load_data_from_csv(file_path)
11
12    # Preprocesa datos
13    X, y, tipos, y_operaciones, tokenizer, max_len, tipo_to_index, operaciones_to_index = preprocess_data(expresiones, etiquetas)
14
15    # Construye el modelo
16    vocab_size = len(tokenizer.get_vocab()) + 1
17    embedding_dim = 64
18    num_types = len(tipo_to_index)
19    num_operaciones = len(operaciones_to_index)
20    model = build_model(vocab_size, embedding_dim, num_types, num_operaciones, max_len)
21
22    # Entrenar el modelo
23    train_model(model, X, {'tipos': y_tipos, 'operaciones': y_operaciones})
24
25    # Hacer una predicción
26    print("Modelo entrenado. Haciendo una predicción...")
27    expresion_test = str(input("Ingresa una expresión algebraica: "))
28    tipo, operacion = predict_expresion(model, tokenizer, max_len, tipo_to_index, operaciones_to_index, expresion_test)
29    print(f"La expresión '{expresion_test}' es clasificada como: (tipo), Operación: (operacion)")
30
31 if __name__ == "__main__":
32     main()
```



Calculadora Algebraica

Cargar Datos

Eliminar Memoria

Calcular



Universidad de Maricao

Facultad de Ciencias Exactas

Departamento de Matemáticas y Física

Centro de Investigación y Desarrollo

Información y Estadística

Calcular

Fecha: 10/01/2020

Horario: 10:00 AM

Alumno: [Nombre]

➤ **¿Qué tipo de entrenamiento usa el programa?**

El programa utiliza el aprendizaje supervisado que es un subconjunto del *machine learning* que consiste en la deducción de información a partir de datos de entrenamiento. Estos datos se clasifican en dos secciones: **datos de entrenamiento y datos de prueba**. Los datos de entrenamiento se utilizan para entrenar a un modelo, y los datos de prueba son los que se usan para determinar la eficacia del modelo creado.

El objetivo del aprendizaje supervisado es crear un programa que sea capaz de resolver cualquier variable de entrada luego de ser sometido a un proceso de entrenamiento, en pocas palabras, es aquel método que se refiere a la generación de modelos para predecir resultados basándose en ejemplos históricos de dichas variables.

Referencias

- ❖ Ceupe (s.f). Aprendizaje supervisado: Qué es, tipos y ejemplo.
<https://www.ceupe.com/blog/aprendizaje-supervisado.html>
- ❖ Hiatoriadelaempresa.com (s.f). ¿Qué es el preprocesamiento de datos y quién lo utiliza? <https://historiadelaempresa.com/que-es-el-preprocesamiento-de-datos>
- ❖ Morales, M. (18 de agosto de 2023). Clasificación de las expresiones algebraicas: ¿Qué es y cómo funcionan? Conaliteg SEP. <https://libros-conaliteg-sep.com.mx/clasificacion-de-expresiones-algebraicas-que-es-y-como-funciona/>
- ❖ Medina, H (s.f). Expresiones algebraicas. Enciclopedia Iberoamericana. <https://enciclopediaiberoamericana.com/expresiones-algebraicas/#:~:text=Una%20expresión%20algebraica%20es%20una%20expresión%20compuesta%20por,y%20división%2C%20además%20de%20la%20potenciación%20y%20radicación.>
- ❖ Zapata, F (28 de abril de 2020). ¿Qué son las expresiones algebraicas y cuáles son más frecuentes? Lifeder. <https://www.lifeder.com/expresiones-algebraicas/>