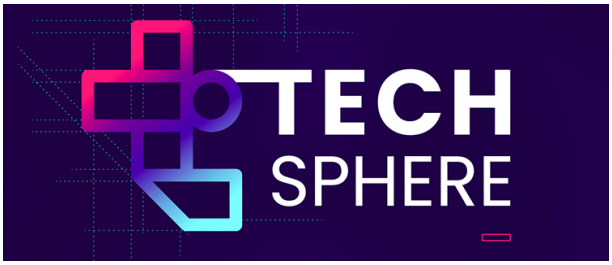

Challenge de Clasificación Biomédica con IA

BioDetectAI

Carlos Alfonso Amariles Vega

Juan Jose Giraldo Muñoz

2025

1 Objetivo

Implementar un sistema de inteligencia artificial que clasifique artículos médicos en uno o varios dominios, utilizando el título y el resumen, con el fin de facilitar la organización y búsqueda de información científica. Track changes are available on all our premium plans, and can be toggled on or off using the option at the top of the Review pane. Track changes allow you to keep track of every change made to the document, along with the person making the change.

2 Metodología

2.1 comprensión del problema

Identificamos la cantidad de registros en el dataset, los tipos de datos disponibles y las etiquetas de dominios médicos. Verificamos la presencia de valores nulos o inconsistencias y observamos la distribución de las etiquetas.

2.2 Calidad del análisis inicial

Relacionando la Regresión con la Clasificación El proyecto de regresión que realizamos (relación entre enfermedades cardiovasculares y neurológicas) y el nuevo reto de clasificación de literatura médica comparten la misma metodología fundamental de la ciencia de datos. Aunque los problemas son diferentes, el flujo de trabajo es idéntico.

2.3 uso de estadisticas

2.4 Planteamiento del reto

El desafío principal es trabajar con datos textuales no estructurados. Los algoritmos de ML/IA requieren representaciones numéricas, por lo tanto, se debe aplicar vectorización y codificación de texto.

2.5 Procesamiento de texto

Realizamos una limpieza de texto pasando los datos a minúsculas, y aplicamos filtrado de stopwords y la representación numérica TF-IDF ya que este último nos permite que los modelos comparen patrones semánticos en forma de matrices numéricas.

2.6 Herramientas/Librerías

Se usaron las siguientes librerías:

- Python: pandas, numpy
- Visualizacion: matplotlib, seaborn
- Herramientas de desarrollo: Google Colab, Git

3 Evidencias

Aquí se muestran algunas capturas del proceso realizado:

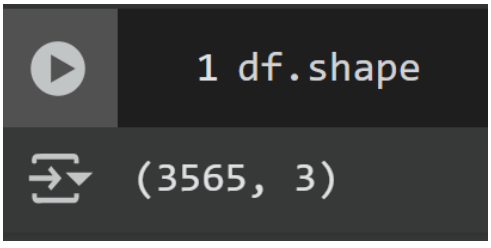

Figure 1: Tamaño del dataset

```
1 print(df['group'].unique())
```

```
['neurological|hepatorenal' 'neurological' 'hepatorenal' 'cardiovascular'
 'neurological|oncological' 'cardiovascular|hepatorenal' 'oncological'
 'neurological|cardiovascular' 'cardiovascular|oncological'
 'neurological|hepatorenal|oncological'
 'neurological|cardiovascular|hepatorenal' 'hepatorenal|oncological'
 'cardiovascular|hepatorenal|oncological'
 'neurological|cardiovascular|hepatorenal|oncological'
 'neurological|cardiovascular|oncological']
```

Figure 2: Verificación de grupos únicos del dataset.

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3565 entries, 0 to 3564
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   title     3565 non-null   object
 1   abstract  3565 non-null   object
 2   group     3565 non-null   object
dtypes: object(3)
memory usage: 83.7+ KB
```

Figure 3: Tipos de datos del dataset.

```
1 print(y_df)
```

```
      cardiovascular  neurological  hepatorenal  oncological
0                  0             1            1            0
1                  0             1            0            0
2                  0             0            1            0
3                  0             1            0            0
4                  0             1            0            0
...              ...           ...          ...          ...
3560               1             1            1            0
3561               0             1            0            0
3562               1             1            0            0
3563               0             1            0            0
3564               1             0            0            1

[3565 rows x 4 columns]
```

Figure 4: Clasificaciones multi-etiquetas del dataset.

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier # Import OneVsRestClassifier
from sklearn.metrics import accuracy_score, classification_report # Also import classification_report

# DEFINIR X e y
X = tfidf_matrix  # Nuestra matriz numérica de TF-IDF
y = y_df  # The binarized labels from earlier in the notebook
#y = df['categoria']  # Las etiquetas que queremos predecir

# Dividir los datos para entrenar y para probar
X_train, X_test, y_train, y_test = train_test_split(X, y.values, test_size=0.25, random_state=42)
```

Figure 5: Division del dataframe en 75 porciento entrenamiento.

```python
# Modelos a probar
models = {
    "Logistic Regression": OneVsRestClassifier(LogisticRegression(max_iter=1000)),
    "Naive Bayes": OneVsRestClassifier(MultinomialNB()),
    "Linear SVM": OneVsRestClassifier(LinearSVC())
}
```

Figure 6: Modelos Implementados de regresión.

Logistic Regression

	precision	recall	f1-score	support
cardiovascular	0.99	0.73	0.84	311
neurological	0.83	0.91	0.87	427
hepatorenal	0.99	0.61	0.75	290
oncological	1.00	0.43	0.61	168
micro avg	0.91	0.72	0.81	1196
macro avg	0.95	0.67	0.77	1196
weighted avg	0.93	0.72	0.80	1196
samples avg	0.90	0.80	0.82	1196

Figure 7: Resultados de la regresión logística.

Naive Bayes

	precision	recall	f1-score	support
cardiovascular	0.99	0.65	0.78	311
neurological	0.77	0.86	0.81	427
hepatorenal	0.99	0.41	0.58	290
oncological	1.00	0.18	0.31	168
micro avg	0.87	0.60	0.71	1196
macro avg	0.94	0.53	0.62	1196
weighted avg	0.91	0.60	0.68	1196
samples avg	0.75	0.67	0.69	1196

Figure 8: Resultados del Naive Bayes.

```
Linear SVM
                precision    recall  f1-score   support

cardiovascular       0.96      0.85      0.90       311
  neurological       0.87      0.91      0.89       427
   hepatorenal       0.99      0.74      0.85       290
   oncological       0.98      0.70      0.82       168

     micro avg       0.93      0.82      0.87      1196
     macro avg       0.95      0.80      0.86      1196
  weighted avg       0.94      0.82      0.87      1196
   samples avg       0.93      0.88      0.89      1196
```

Figure 9: Modelo Linear SMV

```
Resultados comparativos:
                Modelo  F1_macro
0  Logistic Regression  0.766760
1          Naive Bayes  0.623274
2           Linear SVM  0.862115

✅ Mejor modelo: Linear SVM con F1_macro = 0.862115310228444
```

Figure 10: Mejor resultado Comparado de las tres

```
# Define the model
model_nn = Sequential([
    # Input layer: Expects input shapes corresponding to the number of features
    # Use Input layer explicitly if you need more complex models later, otherwise Dense is fine
    # Input(shape=(input_dim,), sparse=True), # Use sparse=True if input is sparse matrix

    # Dense layer 1
    Dense(128, activation='relu', input_shape=(input_dim,)), # Using Dense as the first layer handles input shape

    # Dropout layer for regularization (optional, helps prevent overfitting)
    Dropout(0.2),

    # Dense layer 2 (optional)
    Dense(64, activation='relu'),

    # Dropout layer (optional)
    Dropout(0.2),

    # Output layer: One neuron for each label, with sigmoid activation for multi-label
    Dense(num_labels, activation='sigmoid')
])

# Print the model summary
model_nn.summary()
```

Figure 11: Estructura de la red neuronal

```
# Compile the model
model_nn.compile(optimizer='adam',
                 loss='binary_crossentropy',
                 metrics=['accuracy',
                          tf.keras.metrics.Precision(name='precision'),
                          tf.keras.metrics.Recall(name='recall'),#])
                          tf.keras.metrics.AUC(name='auc')])

print("Modelo compilado con éxito.")
```

Figure 12: Estructura de compilación del modelo Neuronal

```python
from tensorflow.keras.callbacks import EarlyStopping
early_stop = EarlyStopping(
    min_delta=0.001,
    #monitor='val_loss',
    patience=16,
    restore_best_weights=True)
# Define the number of epochs and batch size
epochs = 50  # You can adjust this number
batch_size = 32 # You can adjust this number

# Train the model
history = model_nn.fit(X_train, y_train,
                       epochs=epochs,
                       batch_size=batch_size,
                       validation_data=(X_test, y_test),
                       callbacks=[early_stop],
                       verbose=1) # Set to 0 for less verbose output

print("\nModelo entrenado con éxito.")
```

Figure 13: Configuracion de hiperparametros

```
Classification Report:
                precision    recall  f1-score   support

cardiovascular       0.93      0.87      0.90       311
  neurological       0.88      0.85      0.86       427
   hepatorenal       0.95      0.79      0.86       290
   oncological       0.97      0.70      0.81       168

     micro avg       0.92      0.82      0.87      1196
     macro avg       0.93      0.80      0.86      1196
  weighted avg       0.92      0.82      0.87      1196
   samples avg       0.94      0.88      0.89      1196
```

Figure 14: Resultado de la red Neuronal

4 Resultados obtenidos

Con este trabajo se logró construir un flujo completo de procesamiento de datos, entrenamiento de modelos y validación de resultados, aplicando buenas prácticas en ciencia de datos como la separación de conjuntos de entrenamiento y prueba, el uso de métricas objetivas y la comparación entre diferentes algoritmos.

5 Aporte final

La implementación demuestra que, incluso frente a arquitecturas más complejas, un modelo bien ajustado como el SVM puede ofrecer un rendimiento superior en escenarios específicos, confirmando la importancia de la experimentación y evaluación comparativa antes de seleccionar un modelo final.

6 Conclusiones

Tras la comparación de distintos algoritmos de clasificación, se determinó que el modelo SVM (Support Vector Machine) obtuvo los mejores resultados en términos de precisión y desempeño general, superando a otros métodos probados, incluyendo regresión logística, árboles de decisión e incluso una red neuronal simple.