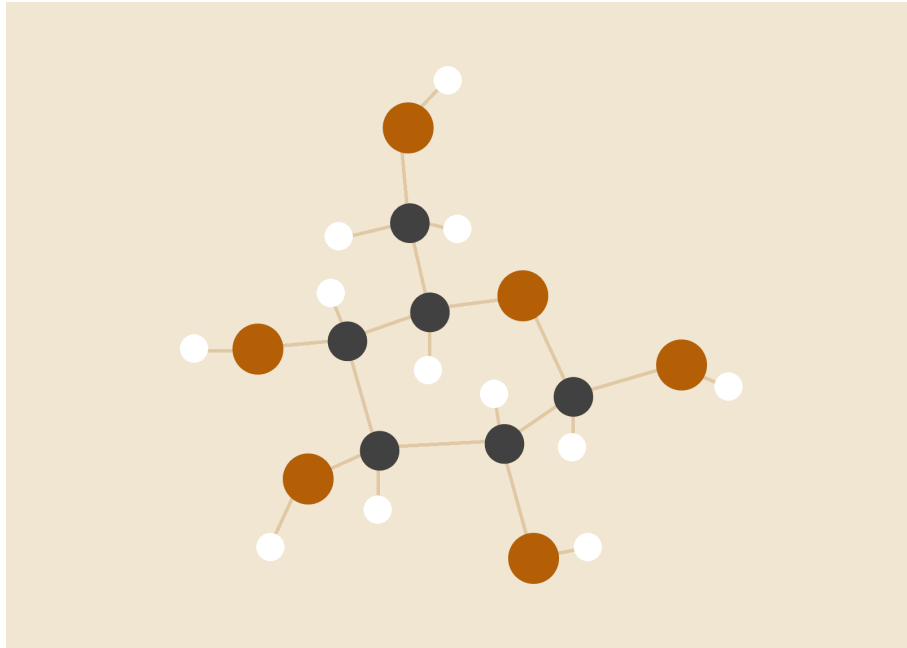


Predicción de Tropos por Medio de Aminoácidos



Dairon Tadeo García Medina

Eduardo Isaí López García

Jesús Yocsan Luévano Flores

Martín Isaí Núñez Villeda

Alan Fernando Martinez Moreno

Pablo David Pérez López

25/11/2024

3^{er} Semestre de Ingeniería en Computación Inteligente

Inteligencia Artificial

Francisco Javier Luna Rosas

Universidad Autónoma de Aguascalientes

INTRODUCCIÓN

El tropismo de la cepa del VIH 1 es muy importante, ya que indican cómo es que “entran” a infectar las células, o sea que se refieren al tipo de receptor celular, así que al lograr identificar que tipo de tropo o tropismo es el VIH, se puede atacar al virus de forma más eficiente. Los tropismos de la cepa VIH 1 son R5, X4, y R5X4

Si bien en la época actual en la que vivimos, con el auge desmedido de la I.A. y su constante y exponencial evolución, ya existen ciertas tecnologías que facilitan aún más trabajos como este, como lo son las “Redes Neuronales Convolucionales”; pero no se usará nada de eso. A base de redes neuronales “simples” o más primitivas, se logra el mismo objetivo, entrenar al programa con cierta cantidad de imágenes para detectar patrones en distintos tipos de caras y así lograr tener una suficiente cantidad de datos de cada emoción para que el programa sea capaz de identificarlos por cuenta propia y en tiempo real.

Si bien la utilidad de este tipo de tecnologías puede no ser tan clara o tan técnica como otras que hemos visto, puede ayudar y servir muy bien para, por ejemplo, reducir el estrés después de un mal día de trabajo; la I.A. al ser capaz de identificar estas emociones, si llegas triste o enojado, podría tener la capacidad de poner en automático música relajante, podría prender la televisión o preparar un café, o si llegas feliz por ejemplo, puede ayudar a mantener el ánimo más tiempo al poner música movida o inclusive con I.A.'s más avanzadas, planear actividades para ese día. Así que efectivamente, puede no ser tan técnicamente práctico, pero puede ser de gran ayuda para reducir los pesos de un día común, o mantener un estado de ánimo positivo, lo que a la larga ayudaría a la persona a estar más tranquila o ser más feliz con su vida cotidiana.

ANÁLISIS

Algoritmos de Detección de Emociones

- **Desempeño General:** Al algoritmo se le pasan una serie de datos, los cuales están clasificados en 3 “tipos” o cepas distintas (R5, X4, y R5X4), estos están divididos en tipos de aminoácidos divididos por proteínas. Cuando se pasan por el modelo, se entrenan para ser capaces, analiza los aminoácidos o proteínas identificadas, dependiendo el caso respectivo, para que logre identificar la cepa, y así, se logre la correcta predicción. Como se puede verificar, el proyecto está realizado en un solo código, el cual a su vez está dividido en varias funciones.
- **Metodología:** Los datos utilizados para esta práctica son datos contenidos en archivos de tipo “.xlsx”, los cuales contienen, como ya se explicó en el punto anterior, contiene los aminoácidos y/o proteínas, las cuales fueron extraídas de varias plataformas de investigación médica-científica (National Library of Medicine, Expasy, etc), con lo cual se obtuvieron los datos de alrededor de 100 aminoácidos, usados para entrenar el modelo.
- **Complejidad:** La complejidad del modelo radica en la dificultad de replicar el comportamiento de una red convolucional utilizando redes neuronales normales, a la par que implementar un clustering de una manera eficiente.
- **Aplicaciones y Limitaciones:** Sus aplicaciones más importantes son, la principal, ayudar a la predicción del tipo de VIH específico, esto para que se pueda mejorar el tratamiento y combate para reducir las consecuencias negativas en la salud que se puedan llegar a tener en una persona. Sin embargo, este algoritmo no resulta tan útil, esto es debido a que la identificación depende de muchos más factores que solo algunos aminoácidos y/o proteínas.

IMPLEMENTACIÓN

Red Neuronal de Predicción de Tropos

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split, KFold
from sklearn.metrics import classification_report, confusion_matrix,
precision_score
from sklearn.cluster import KMeans
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt

# Step 1: Clustering with Unlabeled Data
# Load the original dataset without labels
file_path = 'dataset_2.xlsx'
df_unlabeled = pd.read_excel(file_path)

# Agrupar y preparar las características para el clustering
grouped_unlabeled = df_unlabeled.groupby('Elemento').agg({
    'Count': list,
    'Percentage': list
}).reset_index()

# Create feature vectors by unrolling the lists into columns
dynamically
feature_list_unlabeled = ['Count', 'Percentage']
features_unlabeled = pd.DataFrame({
    f'{col}_{i}': grouped_unlabeled[col].apply(lambda x: x[i] if i <
len(x) else np.nan)
    for col in feature_list_unlabeled
    for i in range(max(grouped_unlabeled[col].apply(len)))
})

# Eliminar columnas vacías
features_unlabeled = features_unlabeled.dropna(axis=1, how='all')
```

```

# Normalize the features
scaler_unlabeled = StandardScaler()
X_unlabeled =
scaler_unlabeled.fit_transform(features_unlabeled.fillna(0))

# KMeans Clustering
k = 3 # Number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
cluster_labels = kmeans.fit_predict(X_unlabeled)

# Add cluster labels to the dataset as an additional feature
features_unlabeled['Cluster'] = cluster_labels

# Assign cluster labels as inferred R5X4 labels
features_unlabeled['Inferred_Type'] =
features_unlabeled['Cluster'].apply(lambda x: 'R5X4' if x == 2 else
('R5' if x == 0 else 'X4'))

# Visualization of Clusters (optional)
plt.scatter(X_unlabeled[:, 0], X_unlabeled[:, 1], c=cluster_labels,
cmap='viridis', s=50)
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[0],
1], s=200, c='red', marker='X')
plt.title("KMeans Clustering")
plt.xlabel("Feature 1 (Count)")
plt.ylabel("Feature 2 (Percentage)")
plt.show()

# Step 2: Supervised Learning with Labeled Data
# Load the datasets for R5 and X4
r5_file_path = 'r5_dataset.xlsx'
x4_file_path = 'x4_dataset.xlsx'

df_r5 = pd.read_excel(r5_file_path, header=None)
df_x4 = pd.read_excel(x4_file_path, header=None)

# Asumir que las columnas están organizadas horizontalmente y necesitan
ser separadas
r5_counts = df_r5.iloc[:, [i for i in range(1, df_r5.shape[1], 3)]]
r5_weights = df_r5.iloc[:, [i for i in range(2, df_r5.shape[1], 3)]]
x4_counts = df_x4.iloc[:, [i for i in range(1, df_x4.shape[1], 3)]]
x4_weights = df_x4.iloc[:, [i for i in range(2, df_x4.shape[1], 3)]]

```

```

# Combine counts and weights into DataFrames
df_r5_combined = pd.concat([r5_counts, r5_weights], axis=1)
df_x4_combined = pd.concat([x4_counts, x4_weights], axis=1)

# Add a column to indicate the virus type
df_r5_combined['Type'] = 'R5'
df_x4_combined['Type'] = 'X4'

# Ensure Type is present in features_unlabeled
features_unlabeled['Type'] = features_unlabeled['Inferred_Type']

# Combine the datasets with inferred R5X4 labels from clustering
df_combined = pd.concat([df_r5_combined, df_x4_combined,
features_unlabeled])

# Convert columns to string to avoid mixed types error
df_combined.columns = df_combined.columns.astype(str)

# Filter for numeric columns and handle missing values
numeric_cols = df_combined.select_dtypes(include=np.number).columns
features_supervised = df_combined[numeric_cols]

# Impute missing values with mean
imputer = SimpleImputer(strategy='mean')
X_supervised = imputer.fit_transform(features_supervised)
y_supervised = df_combined['Type']

# Balance the dataset using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_supervised,
y_supervised)

# One-hot encode the resampled target labels
encoder = OneHotEncoder()
y_resampled = y_resampled.to_numpy() # Convert Series to numpy array
y_encoded_resampled = encoder.fit_transform(y_resampled.reshape(-1,
1)).toarray()

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_encoded_resampled, test_size=0.2, random_state=42)

# Create and compile the neural network model

```

```

model = Sequential([
    Dense(256, input_dim=X_train.shape[1], activation='relu'),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(y_train.shape[1], activation='softmax')
])

optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=200, batch_size=32,
validation_split=0.2)

# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Predict on the test set
y_pred = model.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_test, axis=1)

# Classification report
labels = encoder.categories_[0]
report = classification_report(y_true_labels, y_pred_labels,
labels=np.arange(len(labels)), target_names=labels)
print(report)

# Precision for each class
precision = precision_score(y_true_labels, y_pred_labels, average=None)
for i, label in enumerate(labels):
    print(f"Precision for {label}: {precision[i] * 100:.2f}%")

# Confusion matrix
conf_matrix = confusion_matrix(y_true_labels, y_pred_labels)
print("Confusion Matrix:")
print(conf_matrix)

```

```
# Cross-validation for precision (optional)
def cross_validate_model(X, y, folds=5):
    results = []
    kf = KFold(n_splits=folds)
    for train_idx, test_idx in kf.split(X):
        X_train, X_test = X[train_idx], X[test_idx]
        y_train, y_test = y[train_idx], y[test_idx]

        model.fit(X_train, y_train, epochs=10, batch_size=32,
verbose=0)
        y_pred = np.argmax(model.predict(X_test), axis=1)
        y_true = np.argmax(y_test, axis=1)

        precision = precision_score(y_true, y_pred, average='macro')
        results.append(precision)
    return np.mean(results)

cross_val_precision = cross_validate_model(X_resampled,
y_encoded_resampled)
print(f"Cross-validated Precision: {cross_val_precision * 100:.2f}%")
```

Explicación del Modelo de Predicción:

Inicialización

Se carga el dataset y se estructuran en un DataFrame utilizando la librería **pandas**. Luego se dividen las columnas en variables para predicción y variables objetivo.

Se carga el dataset sin etiquetar desde un archivo Excel. Se agrupan los datos por el elemento (virus) y se agregan las listas de "Count" y "Percentage".

Pre-Procesamiento de Datos

Primero, se normalizan los datos para que estén en un mismo rango, eliminando diferencias; después se codifican las variables categóricas en variables binarias manejables (One-Hot Encoder); finalmente se dividen los datos en datos de entrenamiento y de prueba utilizando la librería **train_test_split**.

Crear Vectores de Características

Se crean vectores de características desplegando las listas en columnas dinámicas y se eliminan las columnas vacías.

Normalizar las Características

Se normalizan las características utilizando `StandardScaler`.

Clustering con KMeans

Se realiza el clustering utilizando KMeans con 3 clusters. Se agregan etiquetas inferidas (R5, X4, R5X4) basadas en los resultados del clustering.

Entrenamiento del Modelo

Se crea una red neuronal con `keras` con 256 nodos ReLu de activación en la capa de entrada, 128 nodos y 64 nodos en capas ocultas, y salidas softmax (se utiliza la función `Dropout` para evitar sobre-ajustes en el modelo). Se compila el modelo y se reajustan los pesos.

Cargar los Datasets Etiquetados

Se cargan los datasets etiquetados desde archivos Excel.

Preparar las Columnas de "Count" y "Weight"

Se separan las columnas de "Count" y "Weight" de los datasets.

Combinar "Count" y "Weight" en DataFrames

Se combinan las columnas de "Count" y "Weight" en DataFrames.

Agregar la Columna de Tipo de Virus

Se agrega una columna para indicar el tipo de virus.

Combinar los Datasets con las Etiquetas Inferidas

Se combinan los datasets etiquetados con los datos inferidos y se convierten los nombres de las columnas a `str`.

Filtrar Columnas Numéricas y Manejar Valores Faltantes

Se filtran las columnas numéricas y se imputan los valores faltantes con la media de la columna.

Balancear el Conjunto de Datos con SMOTE

Se balancea el conjunto de datos utilizando SMOTE.

Codificación One-Hot de las Etiquetas

Se codifican las etiquetas utilizando One-Hot Encoding.

División del Conjunto de Datos en Entrenamiento y Prueba

Se dividen los datos en conjuntos de entrenamiento y prueba.

Crear y Compilar el Modelo

Se crea y compila el modelo de red neuronal.

Entrenar el Modelo

Se entrena el modelo.

Evaluación y Predicción del Modelo

Evaluar el Modelo

Se evalúa la precisión del modelo utilizando la función `evaluate`.

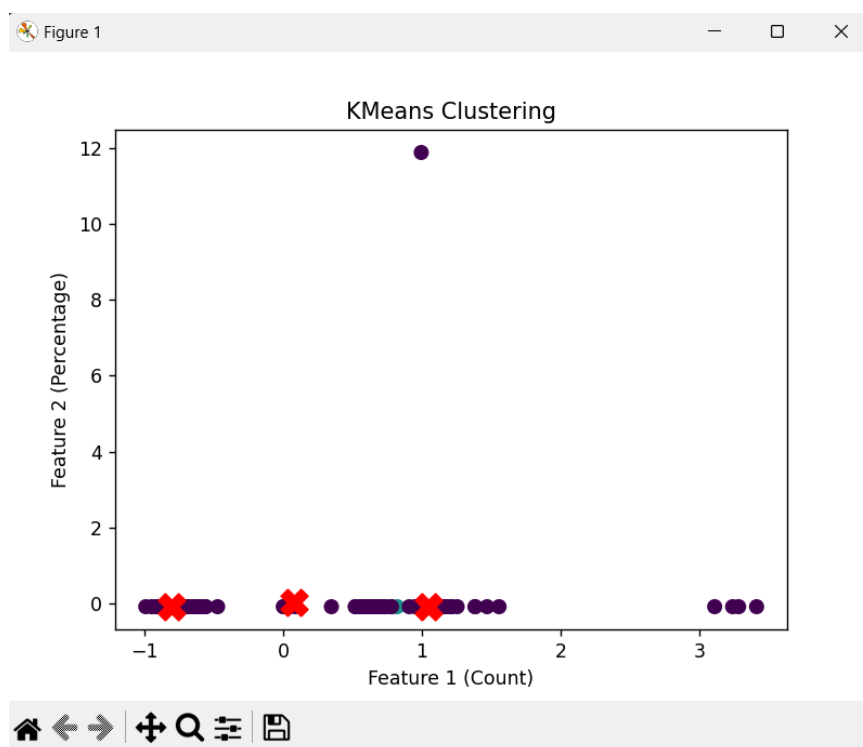
Predicción y Reporte de Clasificación

Se generan las predicciones y las métricas utilizando `predict`, una matriz de confusión, y un reporte con la precisión, el “recall”, y la “F1-Score”.

Validación Cruzada Opcional para Precisión

Como último paso de verificación, se utiliza la validación cruzada para evaluar y promediar el modelo 5 veces y obtener resultados más precisos.

Algunas capturas de evaluación:



R5	1.00	0.68	0.81	31
R5X4	1.00	1.00	1.00	23
X4	0.77	1.00	0.87	34
accuracy			0.89	88
macro avg	0.92	0.89	0.89	88
weighted avg	0.91	0.89	0.88	88

Precision for R5: 100.00%

Precision for R5X4: 100.00%

Precision for X4: 77.27%

Confusion Matrix:

[[21 0 10]

[0 23 0]

[0 0 34]]

3/3  0s 1ms/step


3/3  0s 497us/step

3/3  0s 251us/step

3/3  0s 746us/step

3/3  0s 497us/step

Cross-validated Precision: 93.51%

PS C:\Users\Isai Nuñez\Documents\3er_semestre\Inteligencia_artificial\parcial_III> 

EVALUACIÓN DEL ALGORITMO

Descripción:

El código se basa principalmente en el procesamiento de datos para aplicarlo luego a una red neuronal, de manera que pueda predecir datos, utilizando distintos métodos de entrenamiento y evaluación del modelo, como técnicas de evaluación de precisión, matriz de confusión, o el reporte de clasificación.

Pros:

- Claridad de Datos: Los datos que arroja son claros y fáciles de comprender, de manera que se entiende lo que está pasando.
- Procesamiento: Procesa los datos de forma en que se pueden identificar sesgos o “desviaciones” hacia cierto conjunto de datos.

Contras:

- Volúmenes de datos: Si la cantidad de datos utilizados en el dataset es muy pequeña, puede que el modelo no funcione de forma correcta.
- Temporalidad: Si los datos utilizados o ingresados al modelo son muy distintos en un futuro, hay una gran posibilidad de que el modelo falle.

Complejidad:

- Si bien la presentación de los datos es clara, la interpretación de ciertos datos en específico puede ser algo confusa gracias a la naturaleza de los datos utilizados y el objetivo del modelo.
- La evaluación de datos puede aumentar la complejidad por la utilización de la validación cruzada.

Aplicación:

Al ser un programa extremadamente específico, la única aplicación de este modelo es el objetivo del mismo.

CONCLUSIONES

El proyecto nos ayudó a comprender de mucho mejor forma el funcionamiento de las redes neuronales y manipulación de archivos y datos, cosa que representó un desafío muy grande en cuestión de creación de contenido, ya que el manejar datasets de ese tamaño es demasiado grande.

Una vez teniendo ya los datasets, fue un poco más sencillo el realizar los códigos que se utilizaron, aunque sí se tuvo que modificar de forma en que se pudiesen llegar a los objetivos, lo cual no fue algo demasiado complicado teniendo en cuenta los conocimientos previos.

En conclusión, esta práctica nos ayudó mucho a reafirmar y practicar todo lo que ya hemos visto en el curso del semestre, y representó un reto no solo en lo práctico sino en la organización en general del proyecto, por lo que fue un gran proyecto en cuanto al reto que nos generó.

REFERENCIAS

1. GeeksforGeeks. (2024, 20 marzo). *Clustering in Machine Learning*. GeeksforGeeks.
<https://www.geeksforgeeks.org/clustering-in-machine-learning/>
2. 2.3. *Clustering*. (s. f.). Scikit-learn.
<https://scikit-learn.org/1.5/modules/clustering.html>
3. Regunath, G. (2024, 5 junio). 10 Incredibly Useful Clustering Algorithms — Advancing Analytics. *ADVANCING ANALYTICS*.
<https://www.advancinganalytics.co.uk/blog/2022/6/13/10-incredibly-useful-clustering-algorithms-you-need-to-know>
4. Lamers, S., Salemi, M., McGrath, & Fogel, G. (2008). Prediction of R5, X4, and R5X4 HIV-1 Coreceptor Usage with Evolved Neural Networks. *IEEE/ACM Transactions On Computational Biology And Bioinformatics*, 5(2), 291-300.
<https://doi.org/10.1109/tcbb.2007.1074>