Autor:
Iván García Santillán

Algoritmos predictivos de clasificación: Redes neuronales artificiales

Introducción

Una Red Neuronal Artificial (RNA), o artificial neural network (ANN), es un modelo de aprendizaje automático inspirado en el funcionamiento del cerebro humano. Está compuesta por unidades de procesamiento llamadas neuronas artificiales o nodos, que están organizadas en capas interconectadas. Las redes neuronales artificiales son utilizadas para abordar una amplia variedad de tareas de aprendizaje automático, incluyendo clasificación, regresión, agrupamiento, procesamiento de imágenes, procesamiento de lenguaje natural y más. Por ejemplo, en el campo de la visión por computadora, son esenciales para el reconocimiento de imágenes y rostros, permitiendo innovaciones como los sistemas de seguridad biométrica y las aplicaciones de filtros en redes sociales. En el procesamiento del lenguaje natural, impulsan traductores automáticos y asistentes virtuales, facilitando la comunicación y la interacción con la tecnología. En el sector financiero, estas redes se utilizan para la predicción de series temporales en mercados bursátiles y para la detección de fraudes, analizando patrones complejos en transacciones. En medicina, contribuyen significativamente en el diagnóstico y pronóstico de enfermedades, analizando datos médicos y patrones en imágenes diagnósticas como radiografías y resonancias magnéticas. Además, en el transporte y ámbito del vehículo autónomo, son fundamentales para la interpretación de los datos sensoriales y la toma de decisiones en tiempo real. Aunque también se debe considerar algunas desventajas del algoritmo como la baja velocidad de entrenamiento, uso de memoria considerable, afinamiento moderado-alto requerido y la baja interpretabilidad de sus resultados ya que son considerados como "cajas negras". A pesar de esto, las RNA son muy versátiles para cualquier tipo de aplicación con buenos resultados. Por lo que su uso es muy extendido en el mundo de la

ciencia de datos e IA, considerando que son la base de otros algoritmos más complejos de Deep Learning.

A continuación, se detallan el funcionamiento del algoritmo y un ejemplo en Python, Keras y TensorFlow para la predicción del cáncer de mama utilizando un data set público.

Contenido

Una Red Neuronal Artificial es un modelo de aprendizaje automático inspirado en el funcionamiento del cerebro humano. Algunos conceptos clave relacionados con las redes neuronales artificiales son los siguientes (Berzal, 2019):

Neuronas Artificiales: Cada neurona artificial es una unidad de procesamiento que toma una serie de entradas, las procesa y produce una salida. Las entradas se multiplican por pesos, se suman y se pasa el resultado a una función de activación.
Función de activación: Introduce no linealidad en la red, permitiendo que las redes neuronales aprendan relaciones y patrones complejos en los datos. Existen varias funciones de activación comunes utilizadas en las redes neuronales, entre las que se incluyen: Sigmoide, ReLU, Tangente Hiperbólica, etc. La elección de la función de activación depende del tipo de problema y la arquitectura de la RNA. La función de activación juega un papel fundamental en la capacidad de la RNA para aprender y generalizar a partir de los datos de entrenamiento.
Capas: Las neuronas se organizan en capas. Una red neuronal típica consta de una capa de entrada, una o más capas ocultas y una capa de salida. Las capas ocultas permiten a la red aprender representaciones intermedias y abstraer características de los datos.
Conexiones: Cada neurona en una capa está conectada a todas las neuronas de la capa anterior y a todas las de la capa siguiente. Estas conexiones tienen pesos que se ajustan (aprenden) durante el proceso de entrenamiento.
Entrenamiento: El entrenamiento de una red neuronal implica el ajuste de los pesos de las conexiones para que la red pueda hacer predicciones precisas. Esto se hace utilizando algoritmos de optimización y un conjunto de datos de

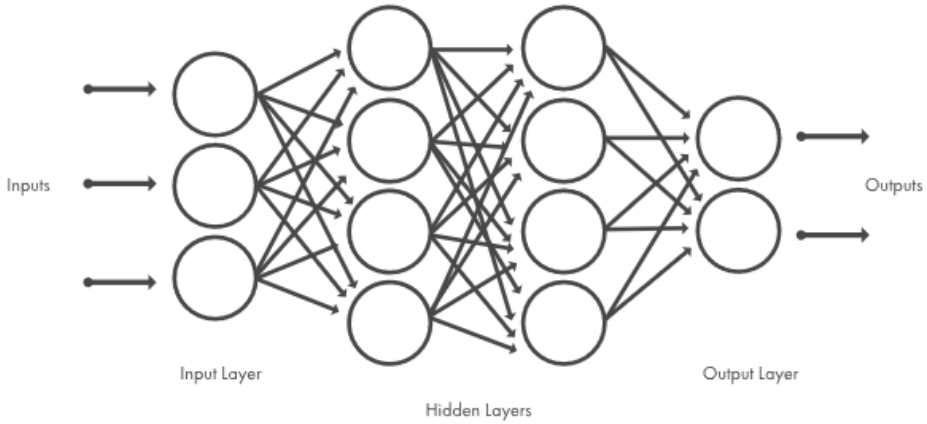
entrenamiento que contiene ejemplos de entrada y las salidas deseadas (etiquetas).

Función de Costo (pérdida): La función de costo mide la discrepancia (error) entre las predicciones de la red y las salidas reales (*ground truth*) en el conjunto de entrenamiento. El objetivo del entrenamiento es minimizar esta función de costo.

Backpropagation: Es un algoritmo utilizado para propagar el error desde la capa de salida hacia atrás a través de la red, ajustando los pesos de las conexiones en función del error cometido.

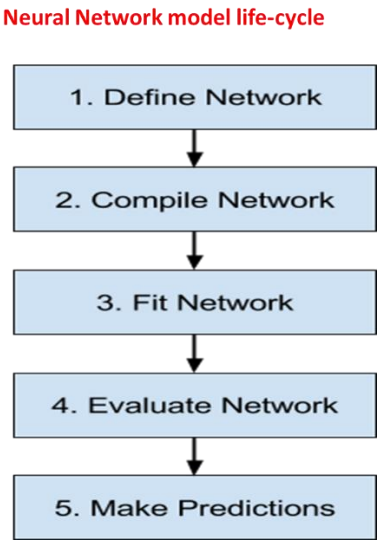
Arquitectura: Las redes neuronales pueden tener diferentes arquitecturas, como redes neuronales feed-forward (Figura 23), redes neuronales recurrentes (RNN) y redes neuronales convolucionales (CNN), cada una adaptada a tareas específicas. En este curso veremos las primeras, conocida como MLP (multi-layer perceptron).

Figura 23. Arquitectura típica de una red neuronal artificial



Las redes neuronales artificiales se han destacado en una amplia gama de aplicaciones, incluyendo el procesamiento de imágenes, el procesamiento de lenguaje natural, la visión por computadora, la traducción automática, el reconocimiento de voz, la recomendación de contenido y muchas otras. Su capacidad para aprender representaciones complejas y realizar tareas sofisticadas las ha convertido en un componente esencial del campo de la inteligencia artificial y el aprendizaje automático. En general, el ciclo de vida de un modelo de red neuronal artificial se muestra en la Figura 24.

Figura 24. Ciclo de vida de un modelo de red neuronal artificial



A continuación, se presenta un ejemplo del algoritmo RNA en Python, Keras y TensorFlow para la predicción del cáncer de mama (benigno=0, maligno=1) utilizando el conjunto de datos Breast Cancer Wisconsin (Diagnostic) disponible en la librería scikit-learn.

```
# RNA for breast cancer

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from keras.models import Sequential
from keras.layers.core import Dense, Dropout
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

# Carga el conjunto de datos Breast Cancer
dataset = load_breast_cancer()
X = dataset.data # 569x30
y = dataset.target # 569x1

# Divide el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Normaliza los datos para que tengan una escala similar
scaler = MinMaxScaler(feature_range=(0,1)) # [0, 1]
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Crea y entrena el modelo RNA
model = Sequential()
model.add(Dense(10, activation='relu', input_dim=30))
model.add(Dropout(0.1))
model.add(Dense(1, activation='sigmoid'))
model.summary()

opt = Adam(lr = 1e-2) # by default lr=1e-3
model.compile(loss='binary_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

# Configurar early stopping para evitar overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
                               restore_best_weights=True)

history = model.fit(X_train, y_train, epochs=100, verbose=1,
              validation_data=(X_test, y_test), callbacks=[early_stopping])

# Realiza predicciones usando el conjunto de prueba
y_pred = model.predict(X_test)

# Convierte las probabilidades en etiquetas binarias (0 o 1)
y_pred = (y_pred > 0.5)

# Muestra el informe de evaluación del modelo entrenado
print(classification_report(y_test, y_pred))

# Matriz de confusión:
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
print("confusion matrix: \n", cm)
# gráfica de la CM
plt.figure(figsize = (8,4))
```

```
sns.heatmap(cm, annot=True, fmt='d')
plt.xlabel('Prediction', fontsize = 12)
plt.ylabel('Real', fontsize = 12)
plt.show()

# Exactitud:
from sklearn.metrics import accuracy_score
acc = accuracy_score(y_test, y_pred)
print("accuracy: ", acc)

# Sensibilidad:
from sklearn.metrics import recall_score
recall = recall_score(y_test, y_pred)
print("recall: ", recall)

# Precisión:
from sklearn.metrics import precision_score
precision = precision_score(y_test, y_pred)
print("precision: ", precision)

# Especificidad
# 'specificity' is just a special case of 'recall'.
# specificity is the recall of the negative class
specificity = recall_score(y_test, y_pred, pos_label=0)
print("specificity: ", specificity)

# Puntuación F1:
from sklearn.metrics import f1_score
f1 = f1_score(y_test, y_pred)
print("f1 score: ", f1)

# Área bajo la curva:
from sklearn.metrics import roc_auc_score
auc = roc_auc_score(y_test, y_pred)
print("auc: ", auc)

# Curva ROC
from sklearn.metrics import roc_curve
plt.figure()
lw = 2
plt.plot(roc_curve(y_test, y_pred)[0], roc_curve(y_test, y_pred)[1], color='darkorange',lw=lw,
label='ROC curve (area = %0.2f)' %auc)
```

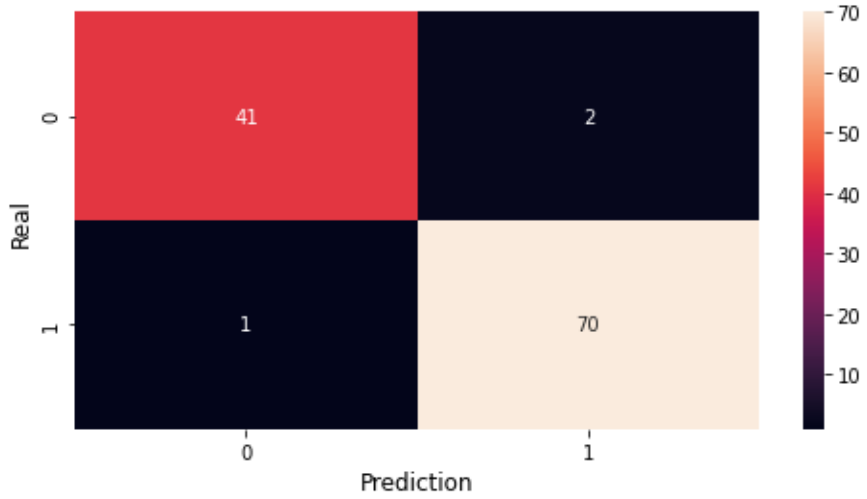
```
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# R Score (R^2 coefficient of determination)
from sklearn.metrics import r2_score
R = r2_score(y_test, y_pred)
print("R2: ", R)

# learning curves
# plot loss during training
plt.title('Loss / binary_crossentropy')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show()
# plot accuracy during training
plt.title('Accuracy')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.legend()
plt.show()

# SHAP (SHapley Additive exPlanations) para explicar las predicciones del modelo.
import shap # pip install shap
# Crear un explainer de SHAP usando en conjunto de entrenamiento
explainer = shap.Explainer(model, X_train)
# Obtener las explicaciones SHAP para el conjunto de prueba
shap_values = explainer.shap_values(X_test)
# Proporciona una visión general de la importancia de las características y su impacto en las
predicciones.
shap.summary_plot(shap_values, X_test, feature_names=dataset.feature_names)
```

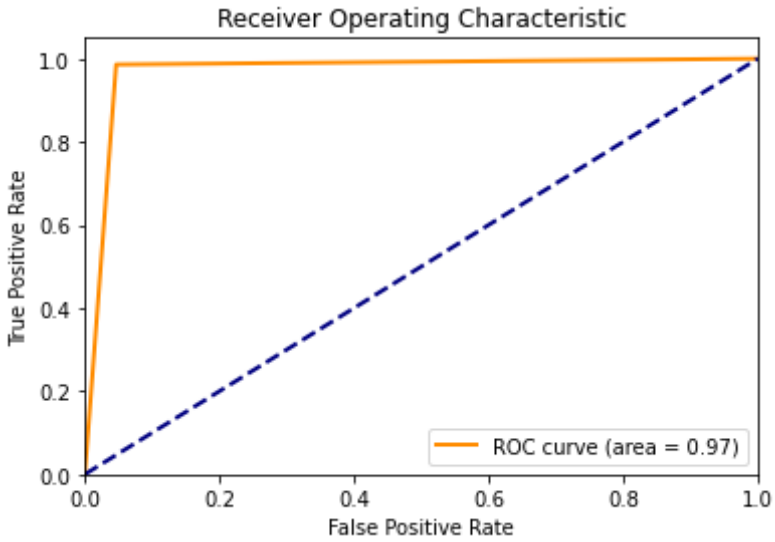
Los resultados del modelo de RNA se presentan a continuación:

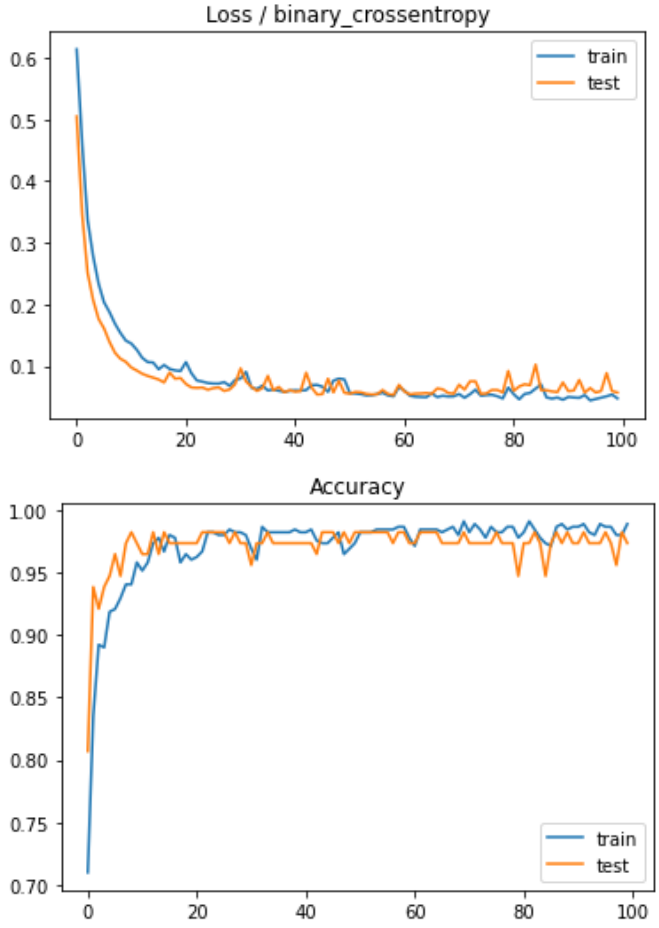
accuracy: 0.97

recall: 0.99

precision: 0.97
specificity: 0.95
f1 score: 0.98
auc: 0.97
R2: 0.89






Loss / binary_crossentropy
train
test
0.6
0.5
0.4
0.3
0.2
0.1
0
20
40
60
80
100
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
0.70
train
test
0
20
40
60
80
100

Las **curvas de aprendizaje** de pérdida (loss) y precisión (accuracy) durante el entrenamiento de un modelo de RNA son útiles para evaluar el desempeño del modelo a lo largo del tiempo y asegurarse de que no esté sobreajustado (overfitting) o subajustado (underfitting), que son dos problemas comunes en el aprendizaje automático que afectan la capacidad de generalización de un modelo.

El **sobreajuste** ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento, capturando no solo las tendencias generales sino también el ruido y las anomalías. Esto hace que el modelo tenga un rendimiento excelente en el conjunto de entrenamiento, pero pobre en datos nuevos (conjunto de prueba). La

causa puede deberse a modelos complejos con demasiados parámetros en comparación con la cantidad de datos disponibles.
El **subajuste** ocurre cuando un modelo es demasiado simple para capturar las tendencias y relaciones subyacentes en los datos. Esto resulta en un rendimiento deficiente tanto en los datos de entrenamiento como en los datos de prueba. La causa poder deberse a modelos demasiado simples con insuficientes parámetros o características relevantes para describir los datos.

Respecto al rendimiento, un clasificador adecuado es aquel que alcanza una exactitud (accuracy) superior al 85%. Por lo que, este modelo de RNA entrenado cumple con este criterio de calidad.
En una RNA, tanto los parámetros como los hiperparámetros juegan roles importantes en la construcción y el ajuste del modelo. Los **parámetros** de las RNA son los valores internos del modelo (pesos y sesgos) que se aprenden durante el entrenamiento.

- **Pesos**: son los coeficientes que multiplican las entradas en cada neurona. Cada conexión entre dos neuronas tiene un peso asociado.
- **Sesgos** (biases): son valores que se suman a las salidas de las neuronas antes de aplicar la función de activación. Desempeñan un papel crucial en la capacidad del modelo para ajustarse y aprender patrones complejos en los datos. Sin sesgos, las RNA serían mucho menos efectivas en aprender y generalizar a partir de los datos. Los pesos y sesgos se ajustan durante el entrenamiento para minimizar la función de pérdida.

En términos matemáticos, para una neurona i en una capa, la salida z_i se calcula como:

$$z_i = \sum_j w_{ij} x_j + b_i$$

donde w_{ij} son los pesos, x_j son las entradas, y b_i es el bias.

Los **Hiperparámetros** de las RNA son configuraciones externas (perillas) establecidas antes del entrenamiento que afectan cómo se entrena el modelo, tales como la arquitectura de la red (# de capas ocultas, # de neuronas por capa), tasa de aprendizaje, número de épocas, tamaño del lote, funciones de activación, método de optimización, etc., las cuales se indican a continuación:
Arquitectura de la Red:

- Número de capas ocultas: La cantidad de capas ocultas en la red.
- Número de neuronas por capa: La cantidad de neuronas en cada capa oculta.

Tasa de Aprendizaje (*Learning Rate*): Controla la magnitud de los ajustes realizados a los pesos y sesgos durante el entrenamiento.
Épocas (Epochs): El número de veces que el algoritmo de entrenamiento recorre todo el conjunto de datos de entrenamiento.
Tamaño del Lote (Batch Size): La cantidad de muestras de entrenamiento que se utilizan para actualizar los pesos en cada paso del entrenamiento.
Funciones de Activación: Las funciones utilizadas para introducir no linealidades en la red. Ejemplos comunes incluyen ReLU (Rectified Linear Unit), sigmoid, Softmax y tanh.
Método de Optimización: El algoritmo utilizado para ajustar los pesos y biases. Ejemplos incluyen SGD (Stochastic Gradient Descent), Adam, RMSprop, etc.
Regularización: Técnicas para prevenir el sobreajuste. Ejemplos incluyen:

- **Dropout**: Probabilidad de que una neurona sea ignorada durante el entrenamiento. Cuando se usa Dropout, no se disminuye el número total de parámetros a entrenar en el modelo. El Dropout simplemente desactiva (o "apaga") temporalmente un subconjunto aleatorio de neuronas durante cada paso de entrenamiento, es decir, en cada actualización de los pesos, lo cual está relacionado con el tamaño del lote (batch size). La desactivación de neuronas es temporal y específica para cada batch. En el siguiente batch, un nuevo conjunto aleatorio de neuronas se desactivará.
- **L1/L2 Regularization**: Penaliza los pesos grandes para evitar el sobreajuste.
- **Inicialización de Pesos**: Estrategia utilizada para asignar los valores iniciales a los pesos. Ejemplos incluyen inicialización aleatoria, Xavier, y He.

La **Parada Temprana** (*early stopping*) también se utiliza durante el entrenamiento de la red neuronal para evitar el sobreajuste. Esta técnica monitorea la pérdida de validación y detiene el entrenamiento si no se observa una mejora después de un número determinado de épocas (*patience*). Esto permite al modelo generalizar mejor a los datos no vistos, evitando un entrenamiento innecesario si el modelo ya no está mejorando, ahorrando tiempo y recursos computacionales. En el ejemplo, si después de 10 épocas (*patience=10*) no hay mejora en la métrica

"val_loss", el entrenamiento se detendrá. La opción "*restore_best_weights=True*" indica que, una vez que se detenga el entrenamiento, los pesos del modelo se restaurarán a los valores que produjeron la mejor métrica en el conjunto de validación durante el entrenamiento.

Adicionalmente, tenga en cuenta que se pueden ajustar varios hiperparámetros del algoritmo (como *activation, loss, optimizer, metrics*, epochs, etc.) para obtener un modelo óptimo. La correcta configuración y ajuste de los hiperparámetros es crucial para el rendimiento de una RNA, y a menudo requiere experimentación y validación exhaustiva. Revise la documentación (ayuda en línea) del algoritmo para más detalles y experimente con ellos.

Utilizando **SHAP (*SHapley Additive exPlanations*)** con modelos de redes neuronales, puedes obtener interpretaciones detalladas de cómo cada característica contribuye a las predicciones del modelo, lo cual es crucial para la transparencia y la comprensión de modelos complejos. Es una herramienta poderosa para entender la importancia de las características y su efecto en las predicciones del modelo. Al observar la dispersión de los valores SHAP y los colores de los puntos, puedes obtener una visión clara de cómo cada característica está influyendo en las predicciones y en qué medida. La distribución de los puntos a lo largo del eje X para cada característica muestra cómo varían los valores SHAP. Una dispersión amplia indica que la característica tiene un impacto variado en las predicciones. Las características del modelo están listadas en el eje Y, donde las características se ordenan por su importancia, de la más importante a la menos importante. El eje X muestra los valores SHAP que indican cuánto cada característica contribuye a la predicción. Un **valor SHAP positivo** significa que la característica aumenta la predicción (clase positiva), mientras que un **valor SHAP negativo** significa que disminuye la predicción. Cada punto de color en el gráfico representa un valor SHAP para una característica de una instancia en el conjunto de datos. El color del punto representa el valor de la característica. Generalmente, los colores van de azul (valor bajo de la característica) a rojo (valor alto de la característica).

Finalmente, aunque la RNA es un modelo fundamental en la caja de herramientas de análisis predictivo, como cualquier otra técnica, presenta tanto ventajas como desventajas que se resumen a continuación:

Ventajas de las redes neuronales artificiales:

Capacidad de aprendizaje y adaptación: Las RNA tienen la capacidad de aprender y modelar relaciones no lineales y complejas, lo que las hace extremadamente potentes para modelar problemas que son difíciles de resolver con técnicas tradicionales. El término "**modelar relaciones no lineales**" se refiere a la capacidad de un modelo estadístico o de aprendizaje automático para capturar y representar relaciones entre variables que no son simplemente proporcionales o directas. En una relación no lineal, los cambios en una variable no producen cambios proporcionales y predecibles en otra variable, lo que significa que la relación entre variables no puede ser descrita adecuadamente por una línea recta, sino por otro tipo de relación como cuadrática, exponencial, logarítmica, sinusoidal, etc.

Generalización: Una vez entrenadas, las redes neuronales son capaces de generalizar a partir de ejemplos no vistos, lo que es crucial para aplicaciones como el reconocimiento de patrones y la clasificación de imágenes.

Manejo de grandes volúmenes de datos: Las RNA son eficaces para manejar grandes cantidades de datos y son capaces de identificar patrones y tendencias que pueden no ser evidentes para los humanos u otros algoritmos.

Flexibilidad: Pueden ser aplicadas a una amplia gama de industrias y problemas, desde el reconocimiento de voz y facial hasta la predicción de series temporales y análisis de sentimientos.

Soporte para diversos tipos de datos: Capaces de procesar datos en diferentes formas, incluyendo imágenes, audio, texto y más, haciendo posible su uso en una variedad de aplicaciones en el mundo real.

Desventajas de las Redes Neuronales Artificiales:

Requieren gran cantidad de datos para el entrenamiento: Para funcionar bien, las RNA generalmente requieren grandes conjuntos de datos de entrenamiento, lo que puede ser un desafío en términos de recopilación y procesamiento de datos.

Costosas computacionalmente: El entrenamiento de modelos de redes neuronales puede ser muy demandante en términos de recursos computacionales, especialmente para redes profundas con muchas capas (Deep Learning). Esto puede requerir hardware especializado, como GPUs, TPUs.

Difíciles de interpretar: A diferencia de otros modelos predictivos (como los árboles de decisión), las RNA son a menudo consideradas como "cajas negras" porque las operaciones internas y la manera en que llegan a una decisión no son fácilmente interpretables.

Sobreajuste: Las redes neuronales son propensas al sobreajuste, especialmente si no tienen suficientes datos de entrenamiento o si la arquitectura de la red es demasiado compleja en relación con la simplicidad del problema.

Sensibilidad a la configuración de los hiperparámetros: La configuración de hiperparámetros (como la tasa de aprendizaje, el número de capas, y el número de nodos por capa) puede tener un gran impacto en el rendimiento del modelo, y encontrar la configuración óptima puede ser un proceso laborioso y técnico.

Vulnerabilidad a ataques adversarios: Las redes neuronales pueden ser susceptibles a ataques adversarios, en los cuales pequeñas pero intencionadas perturbaciones en los datos de entrada pueden engañar a la red para que haga predicciones incorrectas.

Conclusiones

En esta lectura se ha revisado los fundamentos del algoritmo de red neuronal artificial para abordar problemas de clasificación binaria utilizando Python y TensorFlow-Keras. Las redes neuronales ofrecen un poderoso conjunto de capacidades para el modelado de datos complejos y no lineales, pero también

presentan desafíos significativos en términos de necesidades de datos, costos computacionales, y dificultades en la interpretación de sus decisiones. La elección de utilizar redes neuronales debe considerar estos factores en función del problema específico y los recursos disponibles.

Por lo que, se motiva a los estudiantes a revisar material y ejemplos adicionales disponibles libremente en Internet para profundizar en la comprensión del algoritmo y su aplicación eficaz en diferentes problemas de variada complejidad.

Bibliografía:

Berzal, Fernando (2019). Redes neuronales & Deep Learning. USA: Independently published.

Gerón, A. (2023). Aprende Machine Learning con Scikit-Learn, Keras y TensorFlow: Conceptos, herramientas y técnicas para conseguir sistemas inteligentes. Tercera Edición. Anaya Multimedia.

Torres, J. (2020). Python Deep Learning: Introducción práctica con Keras y TensorFlow 2. Marcombo.