

# Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías  
División de Tecnologías para la Integración Ciber-Humana



## Seminario de Solución de Problemas de Inteligencia Artificial II

Ingeniería en Computación  
Sección D05

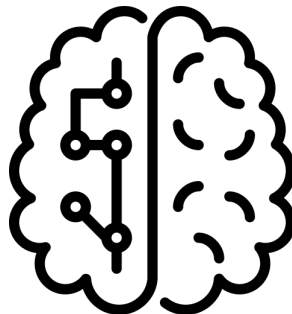
### Practica 1 – Ejercicio 4

Perceptrón Multicapa

**Carrillo Partida Jorge Alberto**

216439258

Jorge.carrillo4392@alumnos.udg.mx



**Profesores**

**M. Diego Alberto Oliva Navarro**

**M. Diego Campos Peña**

22 de Abril de 2024

# Perceptrón Multicapa

## Introducción

Esta práctica se centra en aplicar un perceptrón multicapa para clasificar tres especies de flores del género Iris: setosa, versicolor y virginica. Utilizando un conjunto de datos con características morfológicas de estas flores, se explora cómo las redes neuronales pueden aprender a diferenciar entre estas especies basándose en sus medidas de sépalos y pétalos. Este ejercicio no solo pone en práctica conceptos teóricos de redes neuronales, sino que también resalta la importancia de técnicas de preprocesamiento y validación para asegurar la precisión y robustez del modelo. A través de este estudio, se pretende demostrar la eficacia del MLP en la resolución de problemas de clasificación multicategoría, brindando una comprensión más profunda de su aplicación práctica en el campo de la inteligencia artificial.

## Desarrollo

### Código en Python:

```
# Universidad de Guadalajara
# Centro Universitario de Ciencias Exactas e Ingenierías
# División de Tecnologías para la Integración Ciber-Humana
#
# Seminario de Solución de Problemas de Inteligencia Artificial II
# Ingeniería en Computación
# Sección D05
#
# Práctica 1 – Ejercicio 4
#
# Jorge Alberto Carrillo Partida / 216439258 / jorge.carrillo4392@alumnos.udg.mx
#
# M. Diego Campos – 22 de Abril de 2024
#
# =====
#
# Librerías necesarias para el funcionamiento del programa
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, LeaveOneOut, cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold

# Cargar los datos de los documentos de la práctica
file_path = 'irisbin.csv'
df = pd.read_csv(file_path)

# Transformar las etiquetas a una única etiqueta categórica
df['class'] = df.apply(lambda row: 0 if row['y1'] == -1 and row['y2'] == -1 and row['y3'] == 1 else
                        (1 if row['y1'] == -1 and row['y2'] == 1 and row['y3'] == -1 else 2), axis=1)

# Separar las características y la etiqueta transformada
X = df[['x1', 'x2', 'x3', 'x4']].values
y = df['class'].values

# Escalar las características
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir el dataset en entrenamiento (80%) y prueba (20%)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Crear y entrenar el perceptrón multicapa con parámetros ajustados
mlp = MLPClassifier(hidden_layer_sizes=(12,), activation='relu', solver='adam', max_iter=2000, random_state=42)
mlp.fit(X_train, y_train)

# Evaluar el modelo en el conjunto de prueba
y_pred = mlp.predict(X_test)

# Generar el reporte de clasificación
print("Clasificación en el conjunto de prueba:")
print(classification_report(y_test, y_pred, target_names=['setosa', 'versicolor', 'virginica']))

# Validación Leave-One-Out
loo = LeaveOneOut()
loo_scores = cross_val_score(mlp, X_scaled, y, cv=loo, scoring='accuracy')
loo_mean = np.mean(loo_scores)
loo_std = np.std(loo_scores)

# Resultados de Leave-One-Out
print(f"Leave-One-Out Accuracy: {loo_mean:.2f}")
print(f"Leave-One-Out Std Dev: {loo_std:.2f}")

# Validación Leave-K-Out (Leave-5-Out como ejemplo)
kfold = KFold(n_splits=30)
kfold_scores = cross_val_score(mlp, X_scaled, y, cv=kfold, scoring='accuracy')
kfold_mean = np.mean(kfold_scores)
kfold_std = np.std(kfold_scores)

# Resultados de Leave-K-Out
```

```
print(f"Leave-K-Out Accuracy (k=5): {kfold_mean:.2f}")
print(f"Leave-K-Out Std Dev (k=5): {kfold_std:.2f}")
```

## Código en partes:

### 1. Librerías

```
# Librerías necesarias para el funcionamiento del programa
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, LeaveOneOut, cross_val_score
from sklearn.neural_network import MLPClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold
```

Se importan las librerías necesarias para el funcionamiento del programa. **numpy** se utiliza para trabajar con matrices y realizar operaciones numéricas. **pandas** se emplea para manipular y analizar datos estructurados en forma de dataframes. **train\_test\_split** se utiliza para dividir el conjunto de datos en subconjuntos de entrenamiento y prueba. **LeaveOneOut** y **cross\_val\_score** se emplean para realizar validación cruzada. **MLPClassifier** es el clasificador de perceptrón multicapa de **scikit-learn**. **StandardScaler** se utiliza para escalar las características de los datos, y **classification\_report** se usa para generar un informe de clasificación que incluye métricas como precisión, recall y f1-score. **KFold** se utiliza para realizar la validación cruzada con particiones múltiples.

### 2. Carga de datos, creación de etiquetas y división del dataset

```
# Cargar los datos de los documentos de la práctica
file_path = 'irisbin.csv'
df = pd.read_csv(file_path)

# Transformar las etiquetas a una única etiqueta categórica
df['class'] = df.apply(lambda row: 0 if row['y1'] == -1 and row['y2'] == -1 and row['y3'] == 1 else
                      (1 if row['y1'] == -1 and row['y2'] == 1 and row['y3'] == -1 else 2), axis=1)

# Separar las características y la etiqueta transformada
X = df[['x1', 'x2', 'x3', 'x4']].values
y = df['class'].values

# Escalar las características
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Dividir el dataset en entrenamiento (80%) y prueba (20%)
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

Primero, se carga el dataset **irisbin.csv** en un dataframe **df** usando **pandas**. Luego, se transforma las etiquetas binarias originales en una única etiqueta categórica mediante la función **apply** y una expresión **lambda**. Esta expresión asigna un valor de 0 para **setosa**, 1 para **versicolor** y 2 para **virginica** basándose en los valores de las columnas **y1**, **y2** y **y3**. A continuación, se separan las características (**x1**, **x2**, **x3** y **x4**) en la matriz **X** y las etiquetas en el vector **y**. Las características se escalan usando **StandardScaler** para estandarizarlas, es decir, ajustar cada característica para que tenga una media de 0 y una desviación estándar de 1. Finalmente, se dividen los datos escalados en conjuntos de entrenamiento y prueba en 80% y 20% utilizando **train\_test\_split**.

### 3. Crear, entrenar y evaluar el MLP

```
# Crear y entrenar el perceptrón multicapa con parámetros ajustados
mlp = MLPClassifier(hidden_layer_sizes=(12, 8), activation='relu', solver='adam', max_iter=2000, random_state=42)
mlp.fit(X_train, y_train)

# Evaluar el modelo en el conjunto de prueba
y_pred = mlp.predict(X_test)
```

En esta parte, se crea un perceptrón multicapa con dos capas ocultas que contiene **12** y **8** neuronas, utilizando la función de activación **relu** y el solver **adam** para la optimización. El modelo se entrena con los datos de entrenamiento usando el método **fit**. Después de entrenar el modelo, se realizan predicciones en el conjunto de prueba con el método **predict**, almacenando las predicciones en **y\_pred**.

#### 4. Despliegue de resultados y las validaciones correspondientes

```
# Generar el reporte de clasificación
print("Clasificación en el conjunto de prueba:")
print(classification_report(y_test, y_pred, target_names=['setosa', 'versicolor', 'virginica']))

# Validación Leave-One-Out
loo = LeaveOneOut()
loo_scores = cross_val_score(mlp, X_scaled, y, cv=loo, scoring='accuracy')
loo_mean = np.mean(loo_scores)
loo_std = np.std(loo_scores)

# Resultados de Leave-One-Out
print(f"Leave-One-Out Accuracy: {loo_mean:.2f}")
print(f"Leave-One-Out Std Dev: {loo_std:.2f}")

# Validación Leave-K-Out (Leave-5-Out como ejemplo)
kfold = KFold(n_splits=30)
kfold_scores = cross_val_score(mlp, X_scaled, y, cv=kfold, scoring='accuracy')
kfold_mean = np.mean(kfold_scores)
kfold_std = np.std(kfold_scores)
```

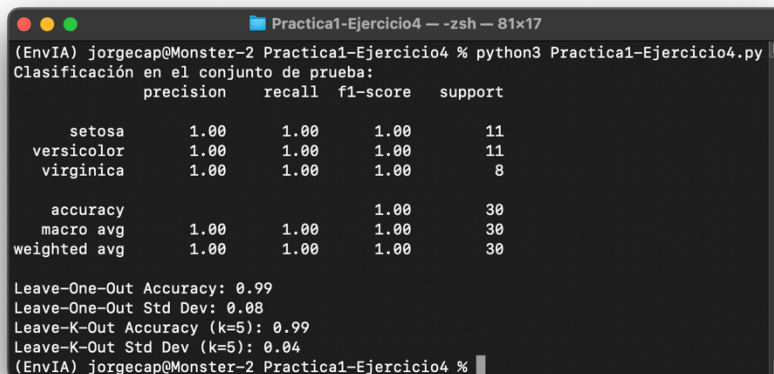
Primero, se genera y muestra un **informe de clasificación** para las predicciones realizadas en el conjunto de prueba. Este informe incluye métricas como la precisión, recall y f1-score para cada clase de la flor Iris (**setosa**, **versicolor** y **virginica**). A continuación, se realiza una validación cruzada **Leave-One-Out**. En este método, cada instancia del dataset se usa una vez como conjunto de prueba y el resto como conjunto de entrenamiento. Se calcula la precisión para cada iteración y se obtienen la media y la desviación estándar de estas precisiones. Estos resultados de **LOO** se imprimen.

Luego, se realiza una validación cruzada **Leave-K-Out** con 30 particiones. En este método, el dataset se divide en 30 subconjuntos y cada uno se usa a su vez como conjunto de prueba mientras los otros 29 se usan para el entrenamiento. Se calcula la precisión para cada iteración y se obtienen la media y la desviación estándar de estas precisiones y estos resultados de **KFold** se imprimen.

#### Resultados

Los resultados de la clasificación en el conjunto de prueba muestran varias métricas importantes. La **precisión** indica la proporción de verdaderos positivos entre los positivos predichos, mientras que el **recall** mide la proporción de verdaderos positivos entre los positivos reales. El **f1-score** es la media armónica de la precisión y el recall, proporcionando una única métrica del rendimiento del modelo. El **support** representa el número de instancias de cada clase en el conjunto de prueba. La **accuracy** global muestra la proporción de predicciones correctas entre todas las predicciones. Las métricas **macro avg** y **weighted avg** son los promedios de las métricas para cada clase. El **macro avg** calcula el promedio de precisión, recall y f1-score sin considerar el número de instancias de cada clase, mientras que el **weighted avg** pondera estas métricas por el número de instancias de cada clase.

En la validación **Leave-One-Out**, cada instancia del dataset se usa una vez como conjunto de prueba y el resto como conjunto de entrenamiento. Los resultados muestran la precisión promedio de todas las iteraciones y la desviación estándar de estas precisiones, proporcionando una medida de la variabilidad del rendimiento del modelo. De manera similar, en la validación **Leave-K-Out** con 30 particiones, el dataset se divide en 30 subconjuntos y cada uno se usa a su vez como conjunto de prueba mientras los otros 29 se usan para el entrenamiento. Los resultados de muestran la precisión promedio y la desviación estándar de todas las iteraciones, ofreciendo otra medida de la consistencia del rendimiento del modelo.



```
Practica1-Ejercicio4 -- -zsh -- 81x17
(EnvIA) jorgecap@Monster-2 Practica1-Ejercicio4 % python3 Practica1-Ejercicio4.py
Clasificación en el conjunto de prueba:
      precision    recall  f1-score   support

   setosa         1.00      1.00      1.00        11
  versicolor     1.00      1.00      1.00        11
   virginica     1.00      1.00      1.00         8

 accuracy         1.00      1.00      1.00        30
  macro avg       1.00      1.00      1.00        30
 weighted avg     1.00      1.00      1.00        30

Leave-One-Out Accuracy: 0.99
Leave-One-Out Std Dev: 0.08
Leave-K-Out Accuracy (k=5): 0.99
Leave-K-Out Std Dev (k=5): 0.04
(EnvIA) jorgecap@Monster-2 Practica1-Ejercicio4 %
```

## **Conclusión**

La implementación y evaluación de un perceptrón multicapa para la clasificación de especies de Iris ha sido una experiencia buena para la comprensión de todo esto. En esta actividad, se ha aprendido la importancia del preprocesamiento de datos, como la transformación de etiquetas y la estandarización de características, para mejorar el rendimiento del modelo. Además, se ha visto cómo técnicas de validación como Leave-One-Out y Leave-K-Out son cruciales para evaluar la consistencia y capacidad de generalización del modelo. Esta actividad ha proporcionado una valiosa oportunidad para aplicar conceptos de redes neuronales en un contexto real, reforzando la comprensión de su funcionamiento y destacando su potencial en la solución de problemas complejos de clasificación. Este ejercicio ha demostrado que, con un adecuado preprocesamiento y validación, los perceptrones multicapa pueden ser herramientas poderosas para la inteligencia artificial.