

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

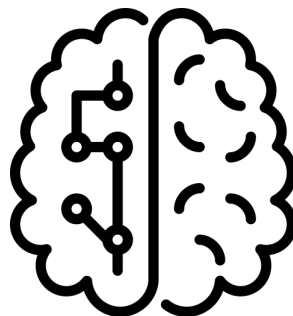
Practica 1 – Ejercicio 2

Perceptrón Simple

Carrillo Partida Jorge Alberto

216439258

Jorge.carrillo4392@alumnos.udg.mx



Profesores

M. Diego Alberto Oliva Navarro

M. Diego Campos Peña

04 de Marzo de 2024

Perceptrón Simple

Introducción

El perceptrón, uno de los modelos más básicos en el campo del aprendizaje automático, actúa como una puerta de entrada fundamental para entender conceptos más avanzados en redes neuronales. En este ejercicio, utilizamos el perceptrón para categorizar diferentes conjuntos de datos, haciendo uso del lenguaje de programación Python y apoyándonos en bibliotecas especializadas como NumPy y pandas. Este método práctico no sólo profundiza nuestra comprensión de los algoritmos de aprendizaje automático, sino que también muestra cómo se pueden aplicar estos conceptos para solucionar problemas prácticos del mundo real.

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 2
#
# Jorge Alberto Carrillo Partida / 216439258 / jorge.carrillo4392@alumnos.udg.mx
#
# M. Diego Campos – 04 de Marzo 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
from sklearn.metrics import accuracy_score

# Clase Perceptron personalizada
class Perceptron:
    def __init__(self, learning_rate=0.1, epochs=10):
        self.learning_rate = learning_rate # Tasa de aprendizaje
        self.epochs = epochs # Número de épocas
        self.weights = None # Inicialización de pesos
        self.bias = 0 # Inicialización de sesgo

    # Función de activación
    def activation(self, x):
        return np.where(x >= 0, 1, -1)

    # Método para entrenar el perceptron
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation(linear_output)
                update = self.learning_rate * (y[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    # Método para realizar predicciones sobre los nuevos datos
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return self.activation(linear_output)

# Función para entrenar y evaluar el perceptrón simple
def train_perceptron(data, n_splits=5, train_size=0.8, learning_rate=0.01, epochs=10):
    results = []
    for _ in range(n_splits):
        # Dividir los datos en entrenamiento y prueba
        X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, :-1], data.iloc[:, -1], train_size=train_size,
            random_state=np.random.randint(1000))

        # Crear y entrenar el perceptrón
        perceptron = Perceptron(learning_rate=learning_rate, epochs=epochs)
        perceptron.fit(X_train.values, y_train.values)

        # Evaluar el perceptrón
        y_pred = perceptron.predict(X_test.values)
        accuracy = accuracy_score(y_test, y_pred)

        # Guardar los resultados
        results.append(accuracy)

    return results

# Cargar los datos de los documentos de la práctica
data_id10 = pd.read_csv('spheres1d10.csv')
```

```

data_2d10 = pd.read_csv('spheres2d10.csv')
data_2d50 = pd.read_csv('spheres2d50.csv')
data_2d70 = pd.read_csv('spheres2d70.csv')

# Entrenar y evaluar el perceptrón personalizado con los datos spheres1d10
accuracies_1d10 = train_perceptron(data_1d10, n_splits=5, learning_rate=0.01, epochs=10)
print(f'Accuracies for spheres1d10.csv: {accuracies_1d10}')

# Entrenar y evaluar el perceptrón personalizado con los datos modificados spheres2d10, spheres2d50 y spheres2d70
accuracies_2d10 = train_perceptron(data_2d10, n_splits=10, learning_rate=0.01, epochs=10)
print(f'Accuracies for spheres2d10.csv: {accuracies_2d10}')

accuracies_2d50 = train_perceptron(data_2d50, n_splits=10, learning_rate=0.01, epochs=10)
print(f'Accuracies for spheres2d50.csv: {accuracies_2d50}')

accuracies_2d70 = train_perceptron(data_2d70, n_splits=10, learning_rate=0.01, epochs=10)
print(f'Accuracies for spheres2d70.csv: {accuracies_2d70}')

```

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
from sklearn.metrics import accuracy_score

```

Este perceptrón se construye y evalúa mediante el uso de la biblioteca **numpy** para cálculos numéricos y **pandas** para la manipulación de datos, junto con herramientas de **scikit-learn** para dividir los datos y calcular la precisión de las predicciones.

2. Perceptron y entrenamiento

```

# Clase Perceptron personalizada
class Perceptron:
    def __init__(self, learning_rate=0.1, epochs=10):
        self.learning_rate = learning_rate # Tasa de aprendizaje
        self.epochs = epochs # Número de épocas
        self.weights = None # Inicialización de pesos
        self.bias = 0 # Inicialización de sesgo

    # Función de activación
    def activation(self, x):
        return np.where(x >= 0, 1, -1)

    # Método para entrenar el perceptron
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self.weights = np.zeros(n_features)
        self.bias = 0

        for _ in range(self.epochs):
            for idx, x_i in enumerate(X):
                linear_output = np.dot(x_i, self.weights) + self.bias
                y_predicted = self.activation(linear_output)
                update = self.learning_rate * (y[idx] - y_predicted)
                self.weights += update * x_i
                self.bias += update

    # Método para realizar predicciones sobre los nuevos datos
    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return self.activation(linear_output)

```

El perceptrón está diseñado como una clase que encapsula la funcionalidad del modelo, incluyendo la inicialización de parámetros, el entrenamiento mediante el ajuste de pesos basado en los datos de entrada, y la capacidad de hacer predicciones sobre nuevos conjuntos de datos. El constructor de la clase **Perceptron** toma dos parámetros: **learning_rate** y el número de épocas, **epochs**, que controlan la rapidez del aprendizaje y el número de veces que el modelo iterará a través del conjunto de datos completo. Dentro de la clase, el método **fit** se utiliza para entrenar el modelo. Aquí, se inicializan los pesos y el sesgo a cero, y luego se ajustan iterativamente a través del algoritmo de entrenamiento del perceptrón. Este algoritmo ajusta los pesos en función de los errores cometidos en las predicciones anteriores, donde el modelo calcula una salida lineal que se pasa a través de una función de activación que transforma el resultado en una etiqueta de clase binaria. El método **predict** se utiliza para evaluar nuevas muestras y predecir su clase basándose en el modelo entrenado. Este método calcula la salida lineal utilizando los pesos y el sesgo ajustados, y luego aplica la función de activación para clasificar la entrada.

3. Evaluación del perceptron

```
# Función para entrenar y evaluar el perceptrón simple
def train_perceptron(data, n_splits=5, train_size=0.8, learning_rate=0.01, epochs=10):
    results = []
    for _ in range(n_splits):
        # Dividir los datos en entrenamiento y prueba
        X_train, X_test, y_train, y_test = train_test_split(data.iloc[:, :-1], data.iloc[:, -1], train_size=train_size,
                                                            random_state=np.random.randint(1000))

        # Crear y entrenar el perceptrón
        perceptron = Perceptron(learning_rate=learning_rate, epochs=epochs)
        perceptron.fit(X_train.values, y_train.values)

        # Evaluar el perceptrón
        y_pred = perceptron.predict(X_test.values)
        accuracy = accuracy_score(y_test, y_pred)

        # Guardar los resultados
        results.append(accuracy)

    return results
```

Para evaluar el perceptrón, se utiliza la función **train_perceptron**. Esta función toma un conjunto de datos y realiza varias divisiones en subconjuntos de entrenamiento y prueba utilizando la función **train_test_split** de **scikit-learn**. Cada partición se utiliza para entrenar el modelo y luego evaluarlo en el conjunto de prueba, calculando la precisión de las predicciones, que es la proporción de etiquetas predichas correctamente respecto a las verdaderas etiquetas de los datos de prueba. Los resultados de estas evaluaciones se almacenan y devuelven como una lista de precisiones para cada partición.

4. Generación de resultados

```
# Cargar los datos de los documentos de la práctica
data_1d10 = pd.read_csv('spheres1d10.csv')
data_2d10 = pd.read_csv('spheres2d10.csv')
data_2d50 = pd.read_csv('spheres2d50.csv')
data_2d70 = pd.read_csv('spheres2d70.csv')

# Entrenar y evaluar el perceptrón personalizado con los datos spheres1d10
accuracies_1d10 = train_perceptron(data_1d10, n_splits=5, learning_rate=0.01, epochs=10)
print(f'Accuracies for spheres1d10.csv: {accuracies_1d10}')

# Entrenar y evaluar el perceptrón personalizado con los datos modificados spheres2d10, spheres2d50 y spheres2d70
accuracies_2d10 = train_perceptron(data_2d10, n_splits=10, learning_rate=0.01, epochs=10)
print(f'Accuracies for spheres2d10.csv: {accuracies_2d10}')

accuracies_2d50 = train_perceptron(data_2d50, n_splits=10, learning_rate=0.01, epochs=10)
print(f'Accuracies for spheres2d50.csv: {accuracies_2d50}')

accuracies_2d70 = train_perceptron(data_2d70, n_splits=10, learning_rate=0.01, epochs=10)
print(f'Accuracies for spheres2d70.csv: {accuracies_2d70}')
```

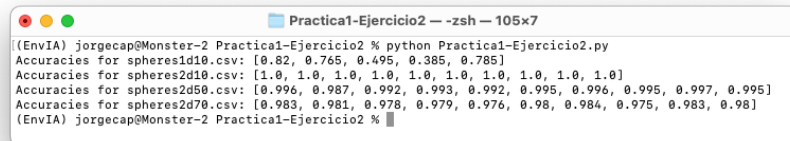
Finalmente, el script carga datos de cuatro archivos CSV diferentes y realiza el entrenamiento y la evaluación del perceptrón para cada uno, imprimiendo las precisiones alcanzadas para cada conjunto de datos.

Resultados

Estos resultados ayudan a verificar la efectividad del perceptrón bajo diferentes condiciones de datos, desde datos ligeramente perturbados hasta aquellos con perturbaciones significativas, como reflejan los diferentes archivos CSV utilizados.

```
Practica1-Ejercicio2 -- zsh -- 105x7
((EnvIA) jorgecap@Monster-2: Practica1-Ejercicio2 % python Practica1-Ejercicio2.py
Accuracies for spheres1d10.csv: [0.725, 0.455, 0.76, 0.76, 0.86]
Accuracies for spheres2d10.csv: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Accuracies for spheres2d50.csv: [0.994, 0.989, 0.997, 0.993, 0.994, 0.995, 0.995, 0.994, 0.991, 0.995]
Accuracies for spheres2d70.csv: [0.986, 0.981, 0.98, 0.957, 0.979, 0.958, 0.974, 0.966, 0.983, 0.98]
((EnvIA) jorgecap@Monster-2: Practica1-Ejercicio2 %
```

```
Practica1-Ejercicio2 -- zsh -- 105x7
Accuracies for spheres2d70.csv: [0.986, 0.981, 0.98, 0.957, 0.979, 0.958, 0.974, 0.966, 0.983, 0.98]
((EnvIA) jorgecap@Monster-2: Practica1-Ejercicio2 % python Practica1-Ejercicio2.py
Accuracies for spheres1d10.csv: [0.5, 0.62, 0.78, 0.89, 0.87]
Accuracies for spheres2d10.csv: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Accuracies for spheres2d50.csv: [0.99, 0.995, 0.994, 0.996, 0.995, 0.993, 0.993, 0.996, 0.996, 0.993]
Accuracies for spheres2d70.csv: [0.972, 0.967, 0.976, 0.975, 0.983, 0.984, 0.975, 0.973, 0.975, 0.98]
((EnvIA) jorgecap@Monster-2: Practica1-Ejercicio2 %
```



```
Practica1-Ejercicio2 — zsh — 105x7
(EnvIA) jorgecap@Monster-2 Practica1-Ejercicio2 % python Practica1-Ejercicio2.py
Accuracies for spheres1d10.csv: [0.82, 0.765, 0.495, 0.385, 0.785]
Accuracies for spheres2d10.csv: [1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]
Accuracies for spheres2d50.csv: [0.996, 0.987, 0.992, 0.993, 0.992, 0.995, 0.996, 0.995, 0.997, 0.995]
Accuracies for spheres2d70.csv: [0.983, 0.981, 0.978, 0.979, 0.976, 0.98, 0.984, 0.975, 0.983, 0.98]
(EnvIA) jorgecap@Monster-2 Practica1-Ejercicio2 %
```

Conclusión

Con el perceptrón simple y el manejo de diversos datasets, hemos podido ver el impacto que tienen los ajustes de los parámetros del modelo en los resultados de la clasificación. Esto nos ha ayudado a establecer una comprensión más compleja que servirá de base para el aprendizaje de modelos más avanzados. Los resultados de esta actividad destacan la capacidad del perceptrón para gestionar distintos grados de complejidad en los datos. Este ejercicio también resalta la importancia de integrar proyectos prácticos en el aprendizaje, lo cual es esencial para aplicar teorías complejas en situaciones reales de manera efectiva.