Universidad Autónoma Metropolitana

Unidad Cuajimalpa

Práctica 3

Introducción a las redes neuronales

Integrantes:

→ Barajas Sánchez Maximiliano 221302628

→ Victoria Nava Natalia 2193034858

Materia: Neurociencias Computacionales y aprendizaje automático

Introducción

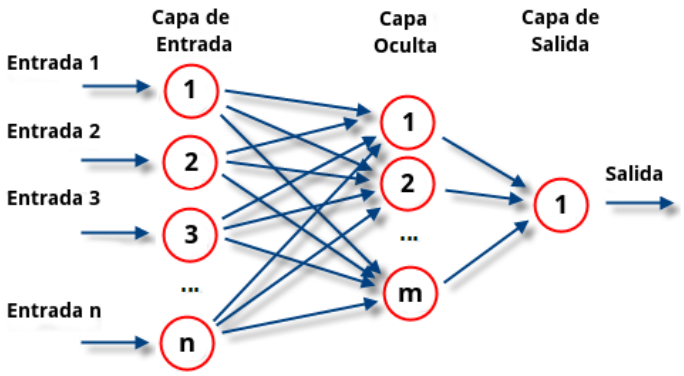
A lo largo de esta práctica se entrenan 2 redes neuronales del tipo “Perceptrón Multicapas para Clasificación con la biblioteca Scikit-Learn de Python”. Además de uso del dispositivo Leap Motion.

Por lo que es importante definir estos conceptos.

Perceptrón multicapa

Es una red neuronal artificial (RNA) formada por múltiples capas, de tal forma que tiene capacidad de resolver problemas que no son linealmente separables.

Genera un modelo predictivo para una o más variables dependientes.



Tiene 3 tipos de capas:

- **Capa de entrada:** Compuesto por las neuronas que introducen los patrones de entrada en la red.
- **Capas ocultas:** Formada por las neuronas de las capas anteriores y cuyas salidas pasan a neuronas de capas posteriores
- **Capas de salida:** Neuronas cuyas salidas corresponden a las salidas de la red.

Para este tipo de red neuronal es muy importante el conjunto de datos. Que se divide en 2 partes:

- **Muestra de entrenamiento:** Corresponde a los registros de datos para entrenar la red neuronal, cierto porcentaje del conjunto de datos debe asignarse a la muestra de entrenamiento.
- **Muestra de prueba:** Es un conjunto independiente de datos que se utilizan para dar seguimiento de los errores durante el entrenamiento.

Scikit-Learn

Es una biblioteca de Python open-source de aprendizaje automático y minería de datos. Ofrece una amplia variedad de herramientas y algoritmos para tareas de clasificación, agrupamiento, selección de características, etc.

Es conocido por su facilidad de uso y documentación detallada.

Leap Motion

Consiste en una pequeña caja rectangular que utiliza una serie de cámaras infrarrojas y sensores para rastrear y capturar los movimientos de las manos y dedos. Enviando los datos a la computadora para procesarlos y utilizarlos con diversos propósitos.

Desarrollo

Actividad 1

A continuación presentamos lo realizado en la actividad 1 con el dataset Iris:

Importamos las bibliotecas y los datos a analizar a lo largo de esta práctica

```
import numpy as np
import sklearn
iris: sklearn.utils._bunch.Bunch= sklearn.datasets.load_iris()
```

Se hizo uso de la biblioteca Sci-kit learn como base de la aplicación del modelo de perceptrón multicapa, a continuación se separaron los datos de tal manera que el 20% de los mismos fueran usados como prueba para el algoritmo y el resto para entrenarlo, tal como establecen los lineamientos del ejercicio:

Separamos los datos en un conjunto de pruebas y un conjunto de entrenamiento

```
datos: np.array=iris.data
etiquetas: np.array=iris.target
```

```
from sklearn.model_selection import train_test_split
datasets: list =train_test_split(datos,etiquetas,test_size=0.2)
```

```
train_data: np.array  ;test_data: np.array;train_labels: np.array ;test_labels:np.array
train_data,test_data,train_labels,test_labels = datasets
```

Después procedimos a instanciar el perceptrón multicapa clasificador de la biblioteca de SciKit Learn:

```
from sklearn.neural_network import MLPClassifier
#Decidimos usar 9 capas ocultas
#El método de optimización Quasi-Newton utilizado fue el método de Broyden - Fletcher - Goldfarb - Shanno de memoria limitada
#El coeficiente de aprendizaje es constante y las iteraciones máximas son 10000
mlp:sklearn.neural_network._multilayer_perceptron.MLPClassifier = MLPClassifier(hidden_layer_sizes=(9), solver='lbfgs',learning_rate='constant', max_iter=10000)
```

Cabe mencionar que se utilizaron 9 capas ocultas, el método de Broyden Fletcher Goldfarb Shanno de memoria limitada como optimizador y un coeficiente de aprendizaje constante junto con un máximo de 10000 iteraciones, decidimos utilizar este optimizador quasi-Newton debido a que la bibliografía establece en gran parte que es más eficiente en cuanto al uso de memoria respecta, una vez entrenado el algoritmo obtuvimos la siguiente salida:

```
mlp.fit(train_data,train_labels)
```

```
MLPClassifier
MLPClassifier(hidden_layer_sizes=9, max_iter=10000, solver='lbfgs')
```

En cuanto a la eficacia de la clasificación del algoritmo sobre los datos de prueba obtuvimos los siguientes resultados:

```
correctos=0
for i in y_estimado==test_labels :
    if i==True:
        correctos+=1
print("El modelo acerto en un ", correctos/len(test_data) *100,"% de los casos")
```

✓ 0.0s

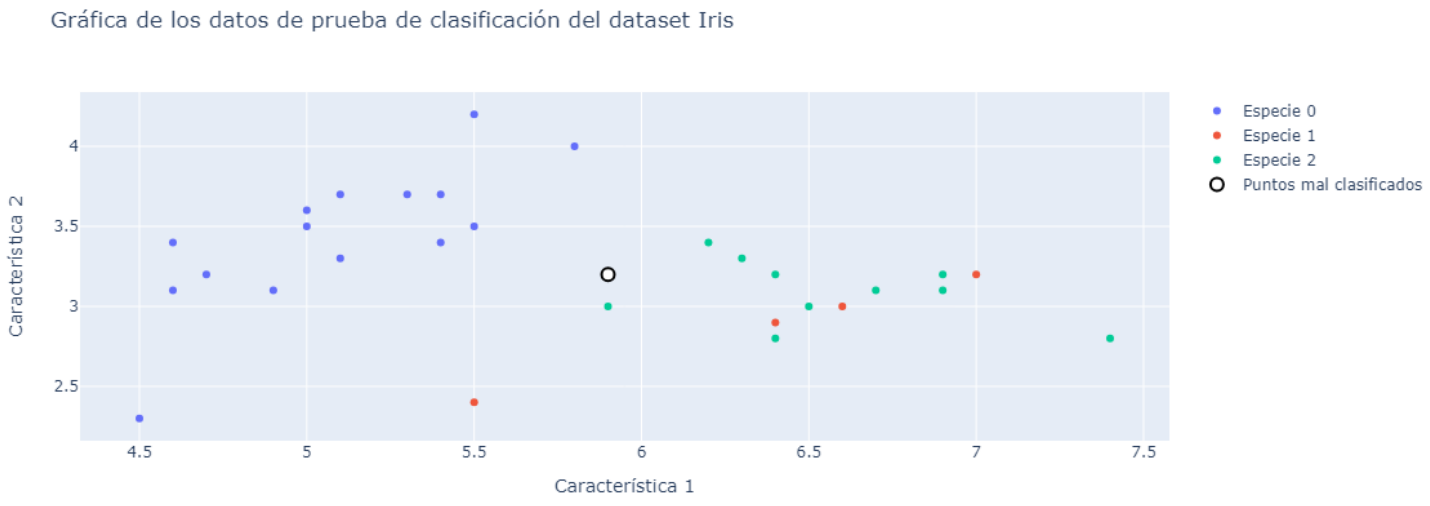
```
El modelo acerto en un  96.66666666666667 % de los casos
```

El modelo tuvo acerto en la etiqueta de los datos de prueba un 96.66% de las veces, en la gráfica podemos observar que solo clásifico erroneamente un solo dato, a continuación mostraremos el código con el cual generamos la gráfica de los datos clasificados por el modelo o bien del conjunto de datos de prueba, se decidió utilizar plotly en lugar de Matplotlib dado que provee gráficas esteticamente mas atractivas e interactivas con el usuario a través de Jupyter Notebook.

```python
import plotly.graph_objects as go
#Buscamos los indices de los puntos bien y mal clasificados
indices_puntos_bien_clasificados = [i for i in range(len(test_labels)) if test_labels[i] == y_estimado[i]]
indices_puntos_mal_clasificados = [i for i in range(len(test_labels)) if test_labels[i] != y_estimado[i]]
#Como menciona la práctica tomamos unicamente 2 características aunque algunas referencias establecen que es mas apropiado usar un PCA
#Guardamos los datos de las plantas correctamente clasificadas por el modelo
x_correctas = [test_data[i][0] for i in indices_puntos_bien_clasificados]
y_correctas = [test_data[i][1] for i in indices_puntos_bien_clasificados]
#Guardamos los de los datos erroneamente clasificados
x_incorrectas = [test_data[i][0] for i in indices_puntos_mal_clasificados]
y_incorrectas = [test_data[i][1] for i in indices_puntos_mal_clasificados]
#Instanciamos el objeto Figure
fig = go.Figure()
# Agregamos los puntos clasificados adecuadamente a la gráfica con un marcador según la clase a la que pertenecen
for label_type in set(test_labels):
    indices = [i for i, etiqueta in enumerate(test_labels) if etiqueta == label_type]
    x = [test_data[i][0] for i in indices]
    y = [test_data[i][1] for i in indices]
    fig.add_trace(go.Scatter(
        x=x,
        y=y,
        mode='markers',
        marker=dict(size=6),
        name=f'Especie {label_type}'
    ))

#Agregamos los puntos erroneamente clasificados con un hueco
fig.add_trace(go.Scatter(
    x=x_incorrectas,
    y=y_incorrectas,
    mode='markers',
    marker=dict(size=10, symbol='circle', line=dict(color='black', width=2),color='white'),
    name='Puntos mal clasificados'
))
fig.update_layout(
    title='Gráfica de los datos de prueba de clasificación del dataset Iris',
    xaxis=dict(title='Característica 1'),
    yaxis=dict(title='Característica 2')
)
#Mostramos la figura
fig.show()
```

La gráfica generada es la siguiente:



Notamos que el algoritmo fracaso en clasificar los elementos del conjunto de prueba una sola vez a lo largo de la clasificación dados los parámetros establecidos por nosotros.

Actividad 2

A lo largo de esta actividad se realizó un entrenamiento de un perceptrón multicapa con ayuda de la herramienta LeapMotion de tal manera que conociendo los gestos, el modelo pudiese interpretar de qué gesto se trata dada una serie de entradas en forma de ángulos respecto a la normal de la palma de la mano, cabe mencionar que se presentaron múltiples dificultades a la hora de trabajar con el código distribuido por el propietario al estar escrito en python 2, a continuación se explicará paso a paso lo realizado a lo largo de este problema.

Inicialmente instalamos los drivers de LeapMotion en Linux (Mint) dado que el paquete públicado en el apartado de la práctica era de terminación .deb , en seguida procedimos a tratar de solucionar el problema de compatibilidad entre Python2 y Python3 dado que la mayoría de bibliotecas como numpy y Sci-Kit Learn ya no ofrecen soporte para esta distribución de Python en particular, decidimos utilizar el mismo enfoque establecido por el profesor de modificar el archivo Sample.py a nuestra conveniencia para poder recopilar los datos de entrenamiento del perceptrón, se realizaron un total de dos cambios mayores al código Sample.py que describiremos a continuación.

Primer Cambio:

En el método on_frame de la clase listener se agregó una lista la cual se encargó de almacenar los ángulos de cada falange de cada dedo respecto a la normal de la mano en cada captura de un gesto individual, para calcular los ángulos guardamos en una tupla las componentes del vector dirección de la respectiva falange y el vector normal, procedimos a calcular el producto punto y después la norma de cada vector con una función adicional euclides, más detalles de la implementación se encuentran en la misma en el archivo con nombre Angulos.py adjunto, a continuación después de capturar los datos de un gesto individual llamamos al destructor de listener de parte de la clase controller de tal manera que podamos introducir los datos de un gesto individual y oprimir enter más tarde para introducir los siguientes mediante el sensor de LeapMotion, para el caso de la primer falange del pulgar (al contar solo con tres falanges) asumimos que el ángulo sería 0 dado que el vector dirección de la falange metacarpal del pulgar es nulo y resulta en norma nula lo que resultaría en una división por 0 a la hora de calcular el ángulo, el ángulo se calcula a través del coseno inverso del producto punto entre el producto de las normas del vector normal y el vector dirección de la falange, a continuación se encuentra el método on_frame modificado:

```
def on_frame(self, controller):
    frame = controller.frame()
    for hand in frame.hands:

        handType = "Left hand" if hand.is_left else "Right hand"
        normal = hand.palm_normal
        direction = hand.direction
        arm = hand.arm
        angulos=[]
        for finger in hand.fingers:
            for b in range(0, 4):
                bone = finger.bone(b)
                normal_tupla=(normal.x,normal.y,normal.z)
                direccionfalange= (bone.direction.x,bone.direction.y,bone.direction.z)
                punto = sum([x*y for x,y in zip(normal_tupla,direccionfalange)])
                if(self.finger_names[finger.type]=="Thumb" and self.bone_names[bone.type] =='Metacarpal' ):
                     angulo = 0
                else:
                     angulo =math.acos(punto/(euclides(normal_tupla) * euclides(direccionfalange)))
                angulos.append(angulo)
        conjunto_datos.append(angulos)
        print angulos
        controller.remove_listener(self)
```

El segundo cambio que realizamos fue con la función main dado que requerimos recopilar los datos para entrenar el perceptrón se optó por guardarlos en un archivo csv de la siguiente manera, de igual forma se imprime el gesto del cuál se está realizando el entrenamiento y entre cada captura de cada gesto individual se requiere de presionar enter de tal suerte que sea más fácil de controlar:

```
def main():
   posiciones=["Piedra" , "Papel" , "Tijeras" ]
   for k in range(3):
      for i in range(40):
      # Create a sample listener and controller
        print "Estas entrenando el"
        print posiciones[k]
        listener = SampleListener()
        controller = Leap.Controller()
        etiquetas.append(k+1)
    # Have the sample listener receive events from the controller
        controller.add_listener(listener)
    # Keep this process running until Enter is pressed
        print "Presiona enter para continuar"
        try:
            sys.stdin.readline()
        except KeyboardInterrupt:
            pass

   with open(file_name, mode='a') as file:
          writer = csv.writer(file)
          for h in range(len(conjunto_datos)):
              writer.writerow(conjunto_datos[h]+[etiquetas[h]])
if __name__ == "__main__":
    main()
```

Ya con los datos en forma de archivo csv pudimos entrenar el perceptrón de manera similar al ejercicio anterior como sigue, el archivo csv se encuentra de igual manera en la carpeta adjunta:

Importamos las librerías a utilizar:

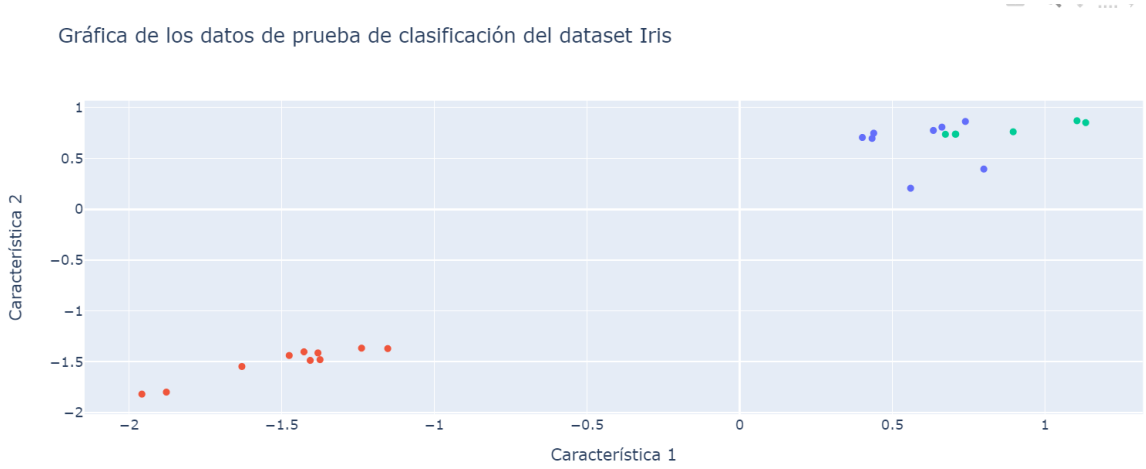
```
import pandas as pd
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from numpy import genfromtxt
my_data = genfromtxt('my_data.csv', delimiter=',')
```

Entrenamos y testeamos el modelo con el conjunto de prueba, en este caso decidimos utilizar 50 capas ocultas, el optimizador adam y un máximo de 1000000 iteraciones:

```
X = my_data[:, 1:-1]  # Tomamos las columnas de significancia estadística
y = my_data[:, -1]   #Tomamos la columna con las etiquetas
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
#Estandarizamos los datos con los que trabajaremos
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
#Instanciamos el Perceptron multicapa, en esta caso decidí usar el método Adam como mi optimizador
mlp = MLPClassifier(hidden_layer_sizes=(50), max_iter=1000000, solver="adam",random_state=42)
#Entrenamos el Clasificador
mlp.fit(X_train, y_train)
#Predecimos las salidas del conjunto de testeo
y_pred = mlp.predict(X_test)
#Evaluamos el modelo
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy*100:.2f}%')
```

No tomamos en cuenta los datos referentes al ángulo existente de la primera falange del pulgar dado que generaba un sesgo considerable en el modelo, en este caso se presentó éxito en la clasificación en el 100% de los casos del conjunto de prueba (esto se encuentra de igual manera en el jupyter notebook, la gráfica de este inciso se encuentra a continuación:



Graficamos nuevamente solo las primeras dos características.

Conclusiones

Después de realizar la práctica anterior podemos concluir que la comprensión del funcionamiento de los modelos de perceptrón como el porqué del uso de los optimizadores es vital para el establecer los parámetros a través de los cuales haremos uso de estos modelos de manera aplicativa en problemas de la vida real, nos enfrentamos a muchas complicaciones a la hora de establecer los parámetros del segundo ejercicio en especial al considerar inicialmente los ángulos nulos dado que resultaba en un fracaso del clasificador bastante considerable.

Referencias Bibliográficas:

de, C. (2005, April 23). *Perceptrón multicapa*. Wikipedia.org; Wikimedia Foundation, Inc. https://es.wikipedia.org/wiki/Perceptr%C3%B3n_multicapa

IBM Documentation. (2023, August 4). Ibm.com. https://www.ibm.com/docs/es/spss-statistics/saas?topic=perceptron-partitions-multilayer