Universidad Autónoma de Querétaro
Facultad de Ingeniería
Ingeniería en Automatización

Optativa profesionalizante II

Clasificador de comandos de voz con el uso de Machine Learning

---

Profesor:
Dr. Mariano
Garduño Aparicio
Mentor:
Ossiel Gastelum

Alumnos:
Martínez Olvera Judith 235363
Fuentes Flores Lorena 263769
Espinosa Bernal Giovanni 262854

Querétaro, Qro. 28 de abril del 2020

Índice

Página

Índice de figuras

1. Introducción

Desde que las primeras computadoras programables fueron concebidas, las personas se preguntaron si tendrían la capacidad de pensar, de aprender y de convertirse en "máquinas inteligentes". El campo de la ciencia que se encarga de resolver este interrogante, se denomina inteligencia artificial. Se trata de un área multidisciplinaria, que a través de ciencias como las ciencias de la computación, la matemática, la lógica y la filosofía, estudia la creación y diseño de sistemas capaces de resolver problemas cotidianos por sí mismos, utilizando como paradigma la inteligencia humana [1]. Para que una maquina pueda comportarse de manera inteligente debería ser capaz de resolver problemas de la manera en que lo hacen los humanos, es decir, en base a la experiencia y el conocimiento [2].
Esto implica que debería ser capaz de modificar su comportamiento en base a cuan precisos son los resultados obtenidos comparados con los esperados. En este sentido podemos encontrar tres grandes grupos de algoritmos de Machine Learning [3]:

- Algoritmos supervisados: estos algoritmos utilizan un conjunto de datos de entrenamiento etiquetados (preclasificados), los cuales procesan para realizar predicciones sobre los mismos, corrigiéndolas cuando son incorrectas. El proceso de entrenamiento continúa hasta que el modelo alcanza un nivel deseado de precisión.
- Algoritmos semi-supervisados: combinan tanto datos etiquetados como no etiquetados para generar una función deseada o clasificador. Este tipo de modelos deben aprender las estructuras para organizar los datos así como también realizar predicciones.
- Algoritmos no supervisados: El conjunto de datos no se encuentra etiquetado y no se tiene un resultado conocido. Por ello deben deducir las estructuras presentes en los datos de entrada, lo puede conseguir a través de un proceso matemático para reducir la redundancia sistemáticamente u organizando los datos por similitud.

Dentro de esta clasificación podemos además encontrar un gran número de algoritmos específicos con diferentes características para el tratamiento de los datos. Entre los más relevantes encontramos:

- Deep Learning: consiste en la utilización de algoritmos para hacer representaciones abstractas de la información y facilitar el aprendizaje automático [4].
- Active Learning: es un caso especial de aprendizaje semi-supervisado donde el algoritmo de aprendizaje puede interactuar con un usuario u otra fuente de información para obtener los resultados deseados [5].

- Support Vector Machines: busca la maximización de la distancia entre la recta o el plano y las muestras que se encuentran a un lado u otro. En el caso que las muestras no sean linealmente separables se utiliza una transformación llamada kernel [6].

Machine Learning es un área de la inteligencia artificial que engloba un conjunto de técnicas que hacen posible el aprendizaje automático a través del entrenamiento con grandes volúmenes de datos. Hoy en día existen diferentes modelos que utilizan esta técnica y consiguen una precisión incluso superior a la de los humanos en las mismas tareas, por ejemplo en el reconocimiento de objetos en una imagen.
La construcción de modelos de Machine Learning requiere adaptaciones propias debido a la naturaleza de los datos o a la problemática a la que se aplica. Así, surge la necesidad de investigar las diferentes técnicas que permitan obtener resultados precisos y confiables en un tiempo razonable.

2. Problemática

Una empresa del sector automotriz se encuentra trabajando en su próximo producto para vehículos inteligentes en el cual necesita implementar un sistema de activación por comandos de voz. El sistema debe ser capaz de identificar los comandos dictados por el usuario y posteriormente ejecutar la instrucción comandada. A través de los comandos se podrá activar cualquier cosa en el vehículo como encender luces, poner seguros, ajustar asiento, abrir cajuela, encender radio, cambiar canción, realizar llamadas, etc.

3. Objetivo

Desarrollar un clasificador de comandos de voz haciendo uso de los algoritmos de Machine Learning.

4. Requisitos

- El clasificador debe identificar por lo menos 10 comandos distintos.
- El clasificador debe funcionar independientemente del timbre de voz.
- El clasificador debe poder ser probado por un usuario. Debe existir un método de entrada de voz para el usuario y el clasificador deberá reaccionar al comando de voz (por ejemplo, mostrar el comando en modo texto).

- Todos los archivos del proyecto deben de estar disponibles en un repositorio de GitHub.

5. Metodología

1. A partir de los requisitos se eligió el lenguaje de programación Python para llevar a cabo el proyecto además de una red neuronal como clasificador. Dado que existen varias partes bien definidas de este proyecto, se hicieron cuatro códigos a fin de hacer más entendibles los diferentes procesos que fueron necesarios para lograr el objetivo establecido. Por el ejemplo el de adquisición de datos; la preparación de una base de datos, la extracción de los datos en una manera entendible para la red neuronal y finalmente la ejecución de la red neuronal que nos muestre los resultados de la clasificación de los comandos.
2. Una vez definidas las partes que formarían el proyecto se realizaron los códigos necesarios para la adquisición de datos: con el código de la sección 10.1 se permite grabar de manera automática cada comando dictado por el usuario y/o agregar un comando si así se requiere. En la sección 10.2 se muestra el código que permite convertir de voz a texto cada audio formando después la base de datos en un archivo .csv que permitirá el entrenamiento de la red neuronal.
3. El código que se encuentra en la sección 10.3 pertenece a la red neuronal utilizada que permite la clasificación del comando que el usuario dicta en tiempo real. Se utilizan 2 capas de neuronas (1 capa oculta) y una bolsa de palabras para organizar los datos de entrenamiento.
4. El programa principal de la sección 10.4 aborda los resultados de la red neuronal y es la sección que se encuentra en contacto con el usuario. En esta parte el usuario dictará el comando que desea o agregar uno nuevo a la lista de comandos que se encuentra en el anexo B.

6. Resultados y discusión

La clasificación de los comandos de voz se logró principalmente por una dedicada investigación sobre las redes neuronales y a que muchas de las fuentes son bastante recientes, es decir, durante el semestre de duración de esta materia. Este clasificador de voz se dividió en cuatro códigos que lo hacen posible y son secuenciales, dicha secuencia se indicó numéricamente en orden ascendente.

```
activar ruido
Reading audio file. Please, wait a moment...
activar fiesta
Reading audio file. Please, wait a moment...
activar acondicionador
Reading audio file. Please, wait a moment...
Activar modo relajación
Reading audio file. Please, wait a moment...
prender aspersores
Reading audio file. Please, wait a moment...
activar encendedor
Reading audio file. Please, wait a moment...
prender luces
Reading audio file. Please, wait a moment...
prender música
Reading audio file. Please, wait a moment...
Apagar modo fiesta
Reading audio file. Please, wait a moment...
Apagar acondicionador
Reading audio file. Please, wait a moment...
Apagar modo relajación
Reading audio file. Please, wait a moment...
Apagar aspersores
Reading audio file. Please, wait a moment...
Apagar encendedor
Reading audio file. Please, wait a moment...
Apagar luces
Reading audio file. Please, wait a moment...
sube la velocidad
Reading audio file. Please, wait a moment...
sube el vidrio
Reading audio file. Please, wait a moment...
aumenta el volumen
Reading audio file. Please, wait a moment...
aumenta la temperatura
Reading audio file. Please, wait a moment...
sube el asiento
```

Figura 1: Resultado del código en python, *Grabación de comandos*.

```
26 documents
26 classes ['baja el asiento', 'Apagar luces', 'Apagar modo fiesta', 'activar encendedor', 'pre
'prender música', 'Apagar modo relajación', 'baja la llanta', 'sube la velocidad', 'prender asp
vidrio', 'baja el vidrio', 'Activar modo relajación', 'activar ruido', 'activar acondicionador'
temperatura', 'baja el volumen', 'disminuye la temperatura', 'disminuye la velocidad', 'aumenta
'activar fiesta', 'Apagar aspersores', 'sube el asiento', 'sube la llanta', 'Apagar acondiciona
encendedor']
24 unique stemmed words ['la', 'vidrio', 'música', 'temperatur', 'apag', 'baj', 'modo', 'activ'
'prend', 'llant', 'fiest', 'volum', 'el', 'encend', 'asiento', 'acondicionad', 'relajación', 'l
'aument', 'aspers', 'velocidad']
# words 24
# classes 26
['activ', 'ruido']
[0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Training with 20 neurons, alpha:0.1, dropout:False
Input matrix: 26x24    Output matrix: 1x26
delta after 10000 iterations:0.004250346588087702
delta after 20000 iterations:0.0028703217390585035
delta after 30000 iterations:0.002294869325251649
delta after 40000 iterations:0.0019616778598720527
delta after 50000 iterations:0.0017384496344866555
delta after 60000 iterations:0.0015758264673406215
delta after 70000 iterations:0.0014507066045335818
delta after 80000 iterations:0.0013506654257103917
delta after 90000 iterations:0.0012683556644788679
delta after 100000 iterations:0.0011991211083237112
saved synapses to: synapses.json
processing time: 6.348582029342651 seconds
```

Figura 2: Resultado del código en python, *Algoritmo: Red Neuronal.*

```
26 documents
26 classes ['baja el asiento', 'Apagar luces', 'Apagar modo fiesta', 'activar encendedor', 'prender
'prender música', 'Apagar modo relajación', 'baja la llanta', 'sube la velocidad', 'prender asperson
vidrio', 'baja el vidrio', 'Activar modo relajación', 'activar ruido', 'activar acondicionador', 'au
temperatura', 'baja el volumen', 'disminuye la temperatura', 'disminuye la velocidad', 'aumenta el v
'activar fiesta', 'Apagar aspersores', 'sube el asiento', 'sube la llanta', 'Apagar acondicionador',
encendedor']
24 unique stemmed words ['la', 'vidrio', 'música', 'temperatur', 'apag', 'baj', 'modo', 'activ', 'ru
'prend', 'llant', 'fiest', 'volum', 'el', 'encend', 'asiento', 'acondicionad', 'relajación', 'luc',
'aument', 'aspers', 'velocidad']
# words 24
# classes 26
['activ', 'ruido']
[0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Training with 20 neurons, alpha:0.1, dropout:False
Input matrix: 26x24    Output matrix: 1x26
delta after 10000 iterations:0.004250346588087702
delta after 20000 iterations:0.0028703217390585035
delta after 30000 iterations:0.002294869325251649
delta after 40000 iterations:0.0019616778598720527
delta after 50000 iterations:0.0017384496344866555
delta after 60000 iterations:0.0015758264673406215
delta after 70000 iterations:0.0014507066045335818
delta after 80000 iterations:0.0013506654257103917
delta after 90000 iterations:0.0012683556644788679
delta after 100000 iterations:0.0011991211083237112
saved synapses to: synapses.json
processing time: 5.756953001022339 seconds
Dictar comando...
Comando capturado
----------------------------------------------------
Espera un momento...
El comando que se capturó fue Apagar luces
----------------------------------------------------
no
found in bag: apag
found in bag: luc
sentence: Apagar luces
 bow: [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
Apagar luces
 classification: [['Apagar luces', 0.9925293579930755]]
```

Figura 3: Resultado del código en python, *Programa principal*.

7. Conclusiones

7.1. Espinosa Bernal Giovanni

La red neuronal como clasificador es muy buena ya que puede tratar con bastantes datos de entrada, además, estos datos no están relacionados entre ellos de alguna manera lineal que pueda ser clasificado con alguna correlación o algún clasificador por agrupamiento ya que son de muchas dimensiones estos datos. A pesar de los nuevos métodos necesitados para la realización de este proyecto, se alcanzó el objetivo que parecía muy difícil al principio pero que gracias a la investigación en el área, pudimos comprender y aplicar. Se trató de automatizar lo más posible de cada proceso ya que Python es un lenguaje de programación que permite el manejo de un macros de manera sencilla por tratarse de un lenguaje de alto nivel. Sin duda alguna es un proyecto mejorable y con muchas aplicaciones, no sólo para el objetivo propuesto en este documento además de que es muy útil en aplicaciones para la industria.

7.2. Fuentes Flores Lorena

El hombre se ha caracterizado siempre por su búsqueda constante de nuevas vías para mejorar sus condiciones de vida, los desarrollos actuales se dirigen al estudio de las capacidades humanas como una fuente de ideas para el diseño de nuevas alternativas para la solución de problemas. Así, la inteligencia artificial es un intento por descubrir y describir aspectos de la inteligencia humana que pueden ser simulados mediante máquinas.

Las redes neuronales son más que otra forma de emular ciertas características propias de los humanos, como la capacidad de memorizar y de asociar hechos, adquiriendo conocimiento a través de la experiencia. Debido a la gran capacidad que tienen estos sistemas fue el método que se optó por utilizar para poder llegar al objetivo propuesto para el proyecto de clasificador de voz.

Si bien es cierto, una parte importante para el buen funcionamiento del algoritmo es la adquisición de datos por lo cual fue el proceso con más detalles pues también fue el que tomó más tiempo. Debido a la dificultad que se presento con el primer planteamiento de analizar las frecuencias de las voces de acuerdo a las palabras que el usuario dictaba, se eligió la alternativa de transformar la voz a texto para que la complejidad del análisis de los datos adquiridos fuera menor. Con ello se tuvo como resultado un porcentaje de confiabilidad muy bueno logrando el objetivo del proyecto.

Debido a las características que el proyecto presenta, se puede tener un gran campo de aplicación para desarrollar diferentes procesos en distintos ámbitos.

7.3. Martínez Olvera Judith

Al inicio del desarrollo de este proyecto se optó por un método más complejo por la razón de que pensamos que lo ideal era obtener características como lo fue la transformada rápida de Fourier o correlación de las grabaciones que tomamos de los comandos, y digo complejo porque fue difícil obtener estos datos y con ellos entrenar una red neurona. El resultados de la primer neuronal fue deficiente puesto que aunque fuera una buena red neural, la base de datos que se obtuvo no fue eficiente.

Al comenzar una nueva etapa del desarrollo de este proyecto, nos dimos cuenta que era más fácil utilizar propiedades que ya tenía el lenguaje de programación en Python y es que, recurrimos al uso de la biblioteca **speech_recognition** que tenía la propiedad de convertir grabaciones de archivos .wav a texto. Con ayuda de lo anterior, nuestra deficiencia que era la base de datos se arregló, puesto que sin importar el timbre de voz o el género de la persona que hablara (e incluso se puede cambiar el idioma español a otro) nuestra base de datos ahora sí no tenía errores.

En conclusión puedo decir que todo dependió de la planeación, ya que en el primer intento nos complicamos mucho en las soluciones propuestas pero al comenzar de nuevo, nos dimos cuenta que era más sencillo de lo que imaginamos, tanto que a la hora de la ejecución; realizar los códigos, grabaciones y base de datos, fue algo muy rápido.

8. Bibliografía

[1] Assessment of the Commercial Applicability of Artificial Intelligence in Electronic Businesses. Thomas Kramer. Diplom.de. 2002.

[2] Data Classification Algorithms and Applications, Charu C. Aggarwal, CRC Press, 2015.

[3] Machine Learning An Algorithmic Perspective Second Edition, Stephen Marsland, CRC Press, 2015.

[4] A Deep Learning. Book in preparation for MIT Press. Bengio, Y., Goodfellow, I. and Courville, USA, 2015.

[5] Active Learning Literature Survey, Settles Burr, Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2014.

[6] A Tutorial on Support Vector Machines for Pattern Recognition, Christopher J.C. Burges, Kluwer Academic Publishers, 1998.

9. Fuentes de consulta

https://machinelearnings.co/text-classification -using-neural-networks-f5cd7b8765c6
https://iamtrask.github.io//2015/07/12/basic-python-network/
https://chatbotslife.com/how-neural-networks-work-ff4c7ad371f7
https://chatbotslife.com/text-classification-using-algorithms-e4d50dcba45
https://iamtrask.github.io//2015/07/27/python-network-part2/
https://code.tutsplus.com/es/tutorials/how-to-read-and-write-csv-files-in-python–cms-29907
https://www.youtube.com/watch?v=q-N6IcgCqCE&t=18s
https://www.youtube.com/watch?v=KcjHfnCteZg

10. Anexo A. Códigos en Python

10.1. Grabación de comandos

```
    # -*- coding: utf-8 -*-
"""
Created on Sat Apr 25 22:48:47 2020

@author: olvju
"""
'''
https://programacionpython80889555.wordpress.com/2018/10/16/
grabacion-de-sonido-con-pyaudio-ejercicio-basico-en-python/
'''
import pyaudio
import wave

FORMAT=pyaudio.paInt16
'''
Formato de las muestras
'''
CHANNELS=2
'''
Número de canales
'''
CHUNK=1024
'''
Descomposición en 1024 frames, para evitar colapso en el procesado de
```

```
la muestra. Unidades de memoria.
'''
RATE=44100
'''
44100 frames por segundo
'''
duracion=4
'''
Tiempo que  durará el archivo en segundos
'''
for i in range(26):
    archivo="grabacion"+str(i+1)+".wav"
    '''
    Asignación del nombre que tendrá el archivo
    '''
    audio=pyaudio.PyAudio()
    '''
    Se crea una nueva instancia de la clase py.Audio
    '''
    stream=audio.open(format=FORMAT,channels=CHANNELS,rate=RATE,
    input=True,frames_per_buffer=CHUNK)
    '''
    Se inicia pyAudio con el método, se empieza la grabación
    '''
    print("grabando audio #"+str(i+1))
    '''
    Permanecerá el mensaje en pantalla durante la ejecución
    del ciclo de lecura y procesamiento de lectura
    '''
    frames=[]

    for i in range(0, int(RATE/CHUNK*duracion)):
        data=stream.read(CHUNK)
        frames.append(data)
    '''
    Ciclo de lectura de sonido
    '''
    print("grabación terminada")
    print("-----------------------------------------------------")
    stream.stop_stream()
```

```
    '''
    Ordena finalizar el procesado de la información
    '''
    stream.close()
    '''
    Cierra el procesado de la información
    '''
    audio.terminate()
    '''
    Finaliza la ejeción del programa
    '''
    waveFile = wave.open(archivo, 'wb')
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(audio.get_sample_size(FORMAT))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b''.join(frames))
    waveFile.close()
    '''
    Se crea un archivo de audio a partir de la información
    recolectada en el ciclo for  anterior
    '''
```

10.2. Conversión de archivos .wav a texto.

```
    # -*- coding: utf-8 -*-
"""
Created on Sun Apr 26 00:47:59 2020

@author: olvju
"""
import speech_recognition as sr
import time

'''
https://www.youtube.com/watch?v=q-N6IcgCqCE&t=18s
'''
comandos = []

for i in range(26):
```

```
    r = sr.Recognizer()
    '''
    Se crea una nueva instancia de la clase Recognizer
    '''

    with sr.AudioFile('C:\\Users\\olvju\\Documents\\8.Optativa
    profesionalizante II\\Clasificador de comandos de voz\\
    grabacion'+str(i+1)+'.wav') as source:
        '''
        Para que el archivo .wav sea la fuente de
        datos que vaya a capturar las palabras a decir se ocupa la clase
        AudioFile
        '''
        audio = r.listen(source)

        '''
        Para capturar el audio se utiliza
        el método listen que está presente
        en la clase Recognizer, esto hasta que
        se detecte un silencio
        '''

        try:
            print("Reading audio file. Please, wait a moment...")
            text = r.recognize_google(audio, language='es-ES')
            comandos.append(text)
            time.sleep(1)
            print(text)

        except:
            print("I am sorry! I can not understand!")
            '''
            El bloque try - except es para controlar
            aquellos casos donde ocurra un error a la hora de
            capturar o de entender que hace el usurio
            '''
archivo="basededatos.csv"
csv=open(archivo,"w")
'''
En la variable archivo se pone el nombre del archivo .csv
```

```
y se abre, la w es para señalar que se va a escribir en él
'''
titulo="Lista de comandos\n"
csv.write(titulo)
'''
Se escribe el título en el archivo .csv
'''
for i in range(26):
    csv.write(comandos[i]+"\n")
```

10.3. Algoritmo: Red Neuronal

```
    # -*- coding: utf-8 -*-
"""
Created on Sat Mar 14 01:25:08 2020

@author: Judith
"""
#https://machinelearnings.co/text-classification-using-neural
-networks-f5cd7b8765c6
#https://github.com/ugik/notebooks/
blob/master/Neural_Network_Classifier.ipynb

# usa el kit de herramientas de lenguaje natural
import nltk
from nltk.stem.lancaster import LancasterStemmer
import os
import json
import datetime
import numpy as np
import time
import csv
stemmer = LancasterStemmer()

# 4 clases de datos de entrenamiento
training_data = [] #datos de entrenamiento
comandos=[]
auxiliar=[]
with open('basededatos.csv') as File:
    reader = csv.reader(File, delimiter=',', quotechar=',',
```

```
                        quoting=csv.QUOTE_MINIMAL)
    for row in reader:
        comandos.append(row)

for i in range(len(comandos)):
    auxiliar.append(comandos[i][0])

training_data=[]
for i in range(len(comandos)-1):
    training_data.append({"class":auxiliar[i+1], "sentence":auxiliar[i+1]})
#print ("%s sentences of training data" % len(training_data))
#----------------------------------------------------------------------
#organizacion de datos y estructuras para trabajar algoritmicamente
words = []
classes = []
documents = []
ignore_words = ['?']
# recorre cada oración en nuestros datos de entrenamiento
for pattern in training_data:
    # # tokenizar/dividir cada palabra en la oración
    w = nltk.word_tokenize(pattern['sentence'])
     # agregar a nuestra lista de palabras, es como un diccionario
    words.extend(w)
     # agregar a documentos en nuestro corpus
     #cada palabra derivada y el número de ocurrencias
    documents.append((w, pattern['class']))
     # agregar a nuestra lista de clases
     #cada clase y la lista de palabras derivadas que contiene
    if pattern['class'] not in classes:
        classes.append(pattern['class'])

# detener y poner en minuscula cada palabra y eliminar duplicados, raíz
words = [stemmer.stem(w.lower()) for w in words if w not in ignore_words]
words = list(set(words))

# eliminar duplicados
classes = list(set(classes))

print (len(documents), "documents")
print (len(classes), "classes", classes)
```

```
print (len(words), "unique stemmed words", words)

#------------------------------------------------------------------
# crea nuestros datos de entrenamiento
training = []
output = []
# crear una matriz vacía para nuestra salida
output_empty = [0] * len(classes)

# conjunto de entrenamiento, bolsa de palabras para cada oración
for doc in documents:
    # inicializar nuestra bolsa de palabras
    bag = []
    # lista de palabras tokenizadas/divididas para el patrón
    pattern_words = doc[0]
     # raiz de  cada palabra
    pattern_words = [stemmer.stem(word.lower()) for word in pattern_words]
    # crear nuestra matriz de bolsa de palabras
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

    training.append(bag)
    # salida es un '0' para cada etiqueta y '1' para la etiqueta actual
    output_row = list(output_empty)
    output_row[classes.index(doc[1])] = 1
    output.append(output_row)

print ("# words", len(words))
print ("# classes", len(classes))
# # sample training/output
i = 0
w = documents[i][0]
#raices de la oración
print ([stemmer.stem(word.lower()) for word in w])
#coincidencia de las raices en la bolsa(1 si es igual)
print (training[i])
#pone un 1 a la clase pertenece
print (output[i])
#------------------------------------------------------------------
# calcular la no linealidad sigmoidea
```

```
def sigmoid(x):
    output = 1/(1+np.exp(-x))
    return output
'''
la salida es la función de sigmoid evaluada en x
'''

# convierte la salida de la función sigmoide a su derivada
def sigmoid_output_to_derivative(output):
    return output*(1-output)

def clean_up_sentence(sentence):
    # tokenizar/dividir el patrón/ divide la oración en palabras
    sentence_words = nltk.word_tokenize(sentence)
    # minusculas de cada palabra
    sentence_words = [stemmer.stem(word.lower()) for word in sentence_words]
    return sentence_words

# conjunto de bolsa de palabras de retorno: 0 o 1 por cada palabra en la bolsa
def bow(sentence, words, show_details=False):
    # tokenize the pattern
    sentence_words = clean_up_sentence(sentence)
    # bolsa de palabras
    bag = [0]*len(words)
    for s in sentence_words:
        for i,w in enumerate(words):
            if w == s:
                bag[i] = 1
                if show_details:
                    print ("found in bag: %s" % w)

    return(np.array(bag))
##********************************THINK
def think(sentence, show_details=False):
    x = bow(sentence.lower(), words, show_details)
    if show_details:
        print ("sentence:", sentence, "\n bow:", x)
    # capa de entrada es nuestra bolsa de palabras
    l0 = x
    # multiplicación matricial de entrada y capa oculta
```

```
    l1 = sigmoid(np.dot(l0,synapse_0))
    # capa de salida
    l2 = sigmoid(np.dot(l1, synapse_1))
    return l2
##################_____________________
#---------------------------------------------------------------------
# función de entrenamiento de redes neuronales para crear pesos sinápticos
/ multiplicación matricial
    # ANN and Gradient Descent code from https://iamtrask.github.io//
    2015/07/27/python-network-part2/
def train(X, y, hidden_neurons=10, alpha=1, epochs=50000, dropout=False,
dropout_percent=0.5):

    print ("Training with %s neurons, alpha:%s, dropout:%s %s" % (hidden_neurons,
    str(alpha), dropout, dropout_percent if dropout else '') )
    #("Entrenamiento con% s neuronas, alfa:% s, abandono:
    print ("Input matrix: %sx%s    Output matrix: %sx%s" %
    (len(X),len(X[0]),1, len(classes)) )
    #("Matriz de entrada:% sx% s Matriz de salida
    np.random.seed(1)

    last_mean_error = 1
     # inicializa aleatoriamente nuestros pesos con media 0
    synapse_0 = 2*np.random.random((len(X[0]), hidden_neurons)) - 1
    synapse_1 = 2*np.random.random((hidden_neurons, len(classes))) - 1

    prev_synapse_0_weight_update = np.zeros_like(synapse_0)
    prev_synapse_1_weight_update = np.zeros_like(synapse_1)

    synapse_0_direction_count = np.zeros_like(synapse_0)
    synapse_1_direction_count = np.zeros_like(synapse_1)

    for j in iter(range(epochs+1)):

       # Avance a través de las capas 0, 1 y 2
        layer_0 = X
        layer_1 = sigmoid(np.dot(layer_0, synapse_0))

        if(dropout):
            layer_1 *= np.random.binomial([np.ones((len(X),hidden_neurons))],
```

```
    1-dropout_percent)[0] * (1.0/(1-dropout_percent))

layer_2 = sigmoid(np.dot(layer_1, synapse_1))

# ¿cuánto perdimos el valor objetivo?
layer_2_error = y - layer_2

if (j% 10000) == 0 and j > 5000:
    # si el error de esta iteración de 10k es mayor que la última
    iteración, explotar
    if np.mean(np.abs(layer_2_error)) < last_mean_error:
        print ("delta after "+str(j)+" iterations:" +
        str(np.mean(np.abs(layer_2_error))) )
        last_mean_error = np.mean(np.abs(layer_2_error))
    else:
        print ("break:", np.mean(np.abs(layer_2_error)), ">",
        last_mean_error )
        break

# ¿en qué dirección está el valor objetivo?
# ¿Estábamos realmente seguros? si es así, no cambies demasiado
layer_2_delta = layer_2_error * sigmoid_output_to_derivative(layer_2)

# ¿Cuánto contribuyó cada valor l1 al error l2 (de acuerdo con
los pesos)?
layer_1_error = layer_2_delta.dot(synapse_1.T)


# ¿en qué dirección está el objetivo l1?
# ¿Estábamos realmente seguros? si es así, no cambies demasiado
layer_1_delta = layer_1_error * sigmoid_output_to_derivative(layer_1)

synapse_1_weight_update = (layer_1.T.dot(layer_2_delta))
synapse_0_weight_update = (layer_0.T.dot(layer_1_delta))

if(j > 0):
    synapse_0_direction_count += np.abs(((synapse_0_weight_update >
    0)+0) - ((prev_synapse_0_weight_update > 0) + 0))
    synapse_1_direction_count += np.abs(((synapse_1_weight_update >
    0)+0) - ((prev_synapse_1_weight_update > 0) + 0))
```

```
        synapse_1 += alpha * synapse_1_weight_update
        synapse_0 += alpha * synapse_0_weight_update

        prev_synapse_0_weight_update = synapse_0_weight_update
        prev_synapse_1_weight_update = synapse_1_weight_update

    now = datetime.datetime.now()

    # persisten las sinapsis
    synapse = {'synapse0': synapse_0.tolist(), 'synapse1': synapse_1.tolist(),
               'datetime': now.strftime("%Y-%m-%d %H:%M"),
               'words': words,
               'classes': classes
              }
    synapse_file = "synapses.json"

    with open(synapse_file, 'w') as outfile:
        json.dump(synapse, outfile, indent=4, sort_keys=True)
    print ("saved synapses to:", synapse_file)
#_-------------------
    ##################____________________
    #construccion del modelo de la red neuronal
X = np.array(training)
y = np.array(output)

start_time = time.time()

train(X, y, hidden_neurons=20, alpha=0.1, epochs=100000,
dropout=False, dropout_percent=0.2)

elapsed_time = time.time() - start_time
print ("processing time:", elapsed_time, "seconds")
#_-------------------
# umbral de probabilidad
ERROR_THRESHOLD = 0.2
# cargar nuestros valores de sinapsis calculados
synapse_file = 'synapses.json'
with open(synapse_file) as data_file:
    synapse = json.load(data_file)
```

```
    synapse_0 = np.asarray(synapse['synapse0'])
    synapse_1 = np.asarray(synapse['synapse1'])

def classify(sentence, show_details=False):
    results = think(sentence, show_details)

    results = [[i,r] for i,r in enumerate(results) if r>ERROR_THRESHOLD ]
    results.sort(key=lambda x: x[1], reverse=True)
    return_results =[[classes[r[0]],r[1]] for r in results]
    print ("%s \n classification: %s" % (sentence, return_results))
    return return_results

#classify("activar vidrio", show_details=True)
#classify("bajar asiento", show_details=True)
```

10.4. Programa principal.

```
    # -*- coding: utf-8 -*-
"""
Created on Sat Apr 25 22:48:47 2020
@author: olvju
"""
'''
https://programacionpython80889555.wordpress.com/2018/10/16/grabacion-
de-sonido-con-pyaudio-ejercicio-basico-en-python/
'''
import pyaudio
import wave
import speech_recognition as sr
import time
import pandas as pd
import TextClassificationNN

comandos = []

def Recepcion_comando():
    FORMAT=pyaudio.paInt16
    CHANNELS=2
    CHUNK=1024
    RATE=44100
```

```
    duracion=4
    archivo="grabacion0.wav"
    audio=pyaudio.PyAudio()
    stream=audio.open(format=FORMAT,channels=CHANNELS,rate=RATE, input=True,
    frames_per_buffer=CHUNK)
    print("Dictar comando...")
    frames=[]
    for i in range(0, int(RATE/CHUNK*duracion)):
        data=stream.read(CHUNK)
        frames.append(data)

    print("Comando capturado")
    print("-----------------------------------------------------")
    stream.stop_stream()
    stream.close()
    audio.terminate()
    waveFile = wave.open(archivo, 'wb')
    waveFile.setnchannels(CHANNELS)
    waveFile.setsampwidth(audio.get_sample_size(FORMAT))
    waveFile.setframerate(RATE)
    waveFile.writeframes(b''.join(frames))
    waveFile.close()

Recepcion_comando()

def Comando_a_texto():
    r = sr.Recognizer()
    with sr.AudioFile('C:\\Users\\olvju\\Documents\\8.Optativa
    profesionalizante II\\Clasificador de comandos de voz\\grabacion0.wav') as sour
        audio = r.listen(source)
        try:
            print("Espera un momento...")
            text = r.recognize_google(audio, language='es-ES')
            comandos.append(text)
            time.sleep(1)
            print("El comando que se capturó fue "+text)
            print("-----------------------------------------------------")

        except:
            print("I am sorry! I can not understand!")
```

```
    return text

auxiliar=Comando_a_texto()

if auxiliar.lower() == "agregar comando":
    print("A continuación podrá dictar el nuevo comando")
    time.sleep(1)
    Recepcion_comando()
    auxiliar=Comando_a_texto()
    comando_agregado =[auxiliar]
    my_df = pd.DataFrame(comando_agregado)
    my_df.to_csv('basededatos.csv', mode='a', index=False,
    header=False,sep=';',decimal=',')
else:
    TextClassificationNN.classify(auxiliar, show_details=True)
```

11. Anexo B. Lista de comandos

- Activar ruido
- Activar fiesta
- Activar acondicionador
- Activar modo relajación
- Prender aspersores
- Activar encendedor
- Prender luces
- Prender música
- Apagar modo fiesta
- Apagar acondicionador
- Apagar modo relajación
- Apagar aspersores
- Apagar encendedor
- Apagar luces

- Sube la velocidad
- Sube el vidrio
- Aumenta el volumen
- Aumenta la temperatura
- Sube el asiento
- Sube la llanta
- Disminuye la velocidad
- Baja el vidrio
- Baja el volumen
- Disminuye la temperatura
- Baja el asiento
- Baja la llanta