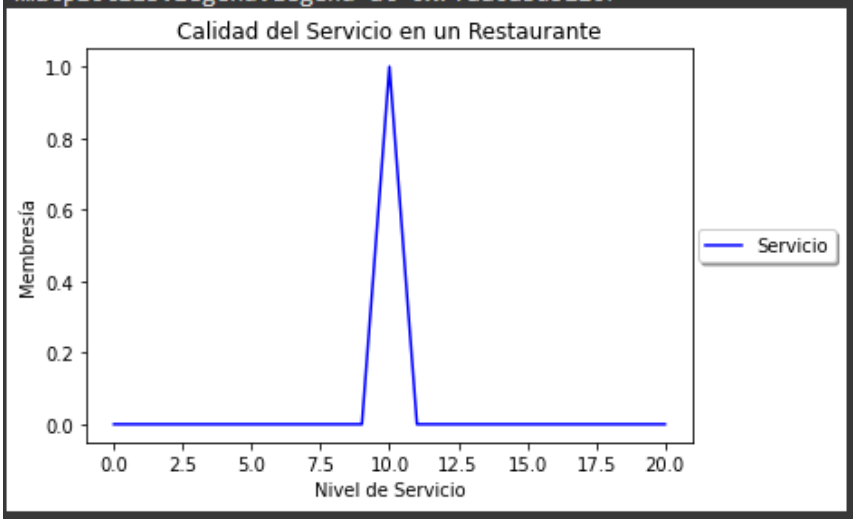
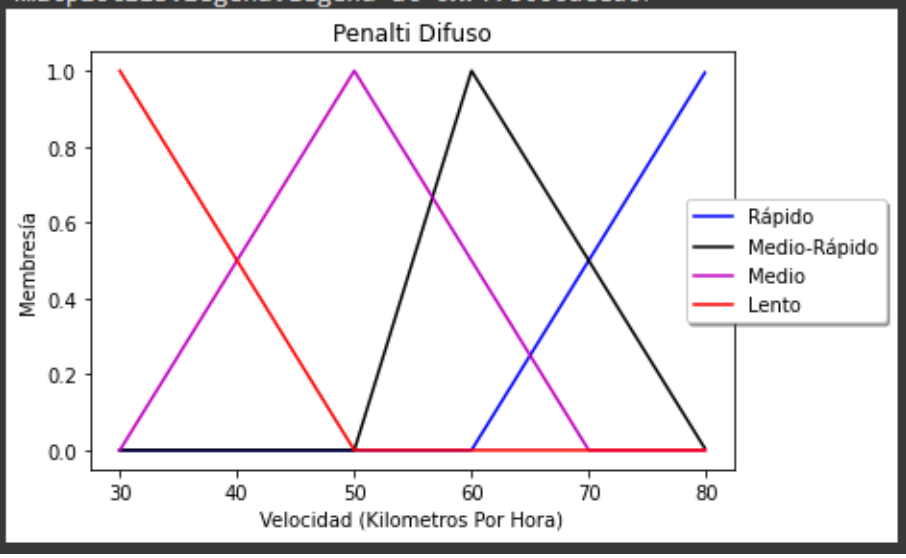
```python
# Función de Membresía Triangular

import numpy as np
import skfuzzy as sk
import matplotlib.pyplot as plt

# Se define un array x para el manejo del factor de calidad en un restaurante
x = np.arange(0, 21, 1)

# Se define un array para la función miembro de tipo triangular
calidad = sk.trimf(x, [10, 10, 10])

# Se grafica la función de membresía
plt.figure()
plt.plot(x, calidad, 'b', linewidth=1.5, label='Servicio')

plt.title('Calidad del Servicio en un Restaurante')
plt.ylabel('Membresía')
plt.xlabel("Nivel de Servicio")
plt.legend(loc='center right', bbox_to_anchor=(1.25, 0.5), ncol=1, fancybox=True, shadow=True)
```

```
<matplotlib.legend.Legend at 0x7faa6abd5110>
```



```python
# Paquetes requeridos
import numpy as np
import skfuzzy as fuzz
import matplotlib.pyplot as plt
%matplotlib inline

# Definiendo los rangos de velocidad de 0 a 80
x = np.arange(30, 80, 0.1)

# Definiendo las funciones miembro triangulares
lento = fuzz.trimf(x, [30, 30, 50])
medio = fuzz.trimf(x, [30, 50, 70])
medio_rapido = fuzz.trimf(x, [50, 60, 80])
rapido = fuzz.trimf(x, [60, 80, 80])

# Dibujando las funciones de membresía
plt.figure()
plt.plot(x, rapido, 'b', linewidth=1.5, label='Rápido')
plt.plot(x, medio_rapido, 'k', linewidth=1.5, label='Medio-Rápido')
plt.plot(x, medio, 'm', linewidth=1.5, label='Medio')
plt.plot(x, lento, 'r', linewidth=1.5, label='Lento')
plt.title('Penalti Difuso')
plt.ylabel('Membresía')
plt.xlabel("Velocidad (Kilometros Por Hora)")
plt.legend(loc='center right', bbox_to_anchor=(1.25, 0.5),
ncol=1, fancybox=True, shadow=True)
```

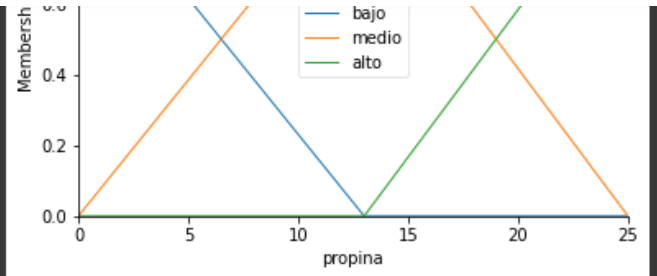
```
<matplotlib.legend.Legend at 0x7f73066dced0>
```

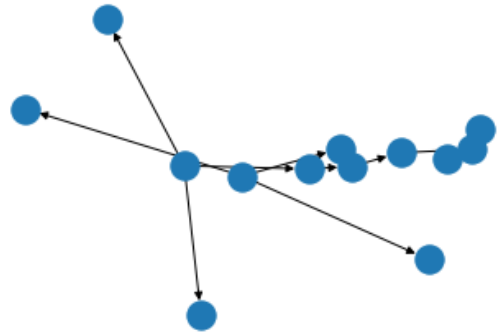


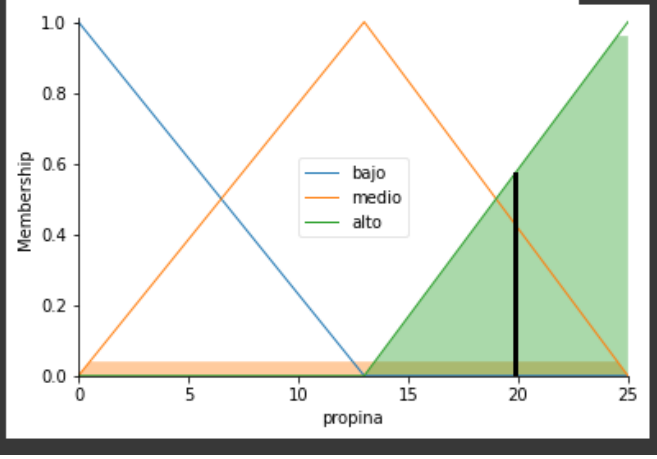
```
[1]  1 pip install -U scikit-fuzzy
```

```
Collecting scikit-fuzzy
  Downloading scikit-fuzzy-0.4.2.tar.gz (993 kB)
     |████████████████████████████████| 993 kB 25.3 MB/s
Requirement already satisfied: numpy>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from scikit-fuzzy) (1.19.5)
Requirement already satisfied: scipy>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from scikit-fuzzy) (1.4.1)
Requirement already satisfied: networkx>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from scikit-fuzzy) (2.6.3)
Building wheels for collected packages: scikit-fuzzy
  Building wheel for scikit-fuzzy (setup.py) ... done
  Created wheel for scikit-fuzzy: filename=scikit_fuzzy-0.4.2-py3-none-any.whl size=894089 sha256=54ccd1312fed87b3bb66927edaa669e93c697ae3a9973015d25968089c942724
  Stored in directory: /root/.cache/pip/wheels/d5/74/fc/38588a3d2e3f34f74588e6daa3aa5b0a322bd6f9420a707131
Successfully built scikit-fuzzy
Installing collected packages: scikit-fuzzy
Successfully installed scikit-fuzzy-0.4.2
```

```python
# CONTROL DIFUSO API

# Elimina las advertencias
import warnings
warnings.filterwarnings('ignore')

# Importa las librerías
import numpy as np
import skfuzzy as fuzz
from skfuzzy import control as ctrl
%matplotlib inline

# Se crean los objetos antecedentes y consecuente a partir de las
# variables del universo y las funciones de membresía
calidad = ctrl.Antecedent(np.arange(0, 11, 1), 'calidad')
servicio = ctrl.Antecedent(np.arange(0, 11, 1), 'servicio')
propina = ctrl.Consequent(np.arange(0, 26, 1), 'propina')

# La población de la función de membresía automática es posible con .automf (3, 5 o 7)
calidad.automf(3)
servicio.automf(3)

# Las funciones de membresía personalizadas se pueden construir interactivamente con la
# API Pythonic
propina['bajo'] = fuzz.trimf(propina.universe, [0, 0, 13])
propina['medio'] = fuzz.trimf(propina.universe, [0, 13, 25])
```







```
model.add(Dense(1, activation='sigmoid'))

# compilar el modelo
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['acc'])

# resumir el modelo
model.summary()

# entrenar el modelo
model.fit(padded_docs, labels, epochs=50, verbose=0)

# evaluar el modelo
loss, accuracy = model.evaluate(padded_docs, labels, verbose=0)
print('Exactitud: %f'% (accuracy*100))
```

```
[[40, 38], [37, 11], [15, 12], [47, 11], [13], [15], [6, 12], [35, 37], [6, 11], [36, 36, 38, 3]]
[[40 38  0  0]
 [37 11  0  0]
 [15 12  0  0]
 [47 11  0  0]
 [13  0  0  0]
 [15  0  0  0]
 [ 6 12  0  0]
 [35 37  0  0]
 [ 6 11  0  0]
 [36 36 38  3]]
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 4, 8)              400
_________________________________________________________________
flatten (Flatten)            (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 433
Trainable params: 433
Non-trainable params: 0
_________________________________________________________________
Exactitud: 89.999998
```

```
[ ]
# Se importan las librerías

import tensorflow as tf
from tensorflow.keras import datasets, layers, models, preprocessing
import tensorflow_datasets as tfds
```

```
[ ]
# Se definen variables de control

max_len = 200
n_words = 10000
dim_embedding = 256
EPOCHS = 20
BATCH_SIZE =500
```

```
[ ]
# Función para la carga de datos

def load_data():
  #load data
  (X_train, y_train), (X_test, y_test) = datasets.imdb.load_data(num_words=n_words)
  # Pad sequences with max_len
  X_train = preprocessing.sequence.pad_sequences(X_train, maxlen=max_len)
  X_test = preprocessing.sequence.pad_sequences(X_test, maxlen=max_len)
  return (X_train, y_train), (X_test, y_test)
```

```
[ ]
# Función para la construcción del modelo

def build_model():
  model = models.Sequential()
  #Input - Emedding Layer
  # the model will take as input an integer matrix of size (batch, input_length)
  # the model will output dimension (input_length, dim_embedding)
    # the largest integer in the input should be no larger
    # than n_words (vocabulary size).
  model.add(layers.Embedding(n_words,
    dim_embedding, input_length=max_len))
```

```
# Se cargan los datos

(X_train, y_train), (X_test, y_test) = load_data()
model=build_model()
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 200, 256)          2560000
_________________________________________________________________
dropout_2 (Dropout)          (None, 200, 256)          0
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 129
=================================================================
Total params: 2,593,025
Trainable params: 2,593,025
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 4/20
50/50 [==============================] - 24s 475ms/step - loss: 0.2196 - accuracy: 0.9143 - val_loss: 0.2934 - val_accuracy: 0.8788
Epoch 5/20
50/50 [==============================] - 24s 475ms/step - loss: 0.1739 - accuracy: 0.9373 - val_loss: 0.2888 - val_accuracy: 0.8762
Epoch 6/20
50/50 [==============================] - 24s 473ms/step - loss: 0.1354 - accuracy: 0.9542 - val_loss: 0.2943 - val_accuracy: 0.8737
Epoch 7/20
50/50 [==============================] - 24s 475ms/step - loss: 0.1052 - accuracy: 0.9672 - val_loss: 0.3052 - val_accuracy: 0.8684
Epoch 8/20
50/50 [==============================] - 24s 473ms/step - loss: 0.0800 - accuracy: 0.9776 - val_loss: 0.3185 - val_accuracy: 0.8666
Epoch 9/20
50/50 [==============================] - 24s 477ms/step - loss: 0.0587 - accuracy: 0.9856 - val_loss: 0.3337 - val_accuracy: 0.8633
Epoch 10/20
50/50 [==============================] - 24s 475ms/step - loss: 0.0443 - accuracy: 0.9900 - val_loss: 0.3552 - val_accuracy: 0.8584
Epoch 11/20
50/50 [==============================] - 24s 476ms/step - loss: 0.0344 - accuracy: 0.9926 - val_loss: 0.3699 - val_accuracy: 0.8577
Epoch 12/20
50/50 [==============================] - 24s 476ms/step - loss: 0.0258 - accuracy: 0.9944 - val_loss: 0.3867 - val_accuracy: 0.8558
Epoch 13/20
50/50 [==============================] - 24s 472ms/step - loss: 0.0194 - accuracy: 0.9967 - val_loss: 0.4163 - val_accuracy: 0.8522
Epoch 14/20
50/50 [==============================] - 24s 477ms/step - loss: 0.0166 - accuracy: 0.9972 - val_loss: 0.4214 - val_accuracy: 0.8538
Epoch 15/20
50/50 [==============================] - 24s 475ms/step - loss: 0.0132 - accuracy: 0.9979 - val_loss: 0.4359 - val_accuracy: 0.8515
Epoch 16/20
50/50 [==============================] - 24s 472ms/step - loss: 0.0110 - accuracy: 0.9982 - val_loss: 0.4541 - val_accuracy: 0.8498
Epoch 17/20
50/50 [==============================] - 24s 476ms/step - loss: 0.0095 - accuracy: 0.9988 - val_loss: 0.4713 - val_accuracy: 0.8497
Epoch 18/20
50/50 [==============================] - 24s 477ms/step - loss: 0.0076 - accuracy: 0.9991 - val_loss: 0.4785 - val_accuracy: 0.8498
Epoch 19/20
50/50 [==============================] - 24s 477ms/step - loss: 0.0074 - accuracy: 0.9988 - val_loss: 0.4919 - val_accuracy: 0.8489
Epoch 20/20
50/50 [==============================] - 24s 476ms/step - loss: 0.0059 - accuracy: 0.9991 - val_loss: 0.4986 - val_accuracy: 0.8506
```

```python
# Se evalúa el modelo

score = model.evaluate(X_test, y_test, batch_size=BATCH_SIZE)
print("\nTest score:", score[0])
print('Test accuracy:', score[1])

```

```
50/50 [==============================] - 2s 45ms/step - loss: 0.4986 - accuracy: 0.8506

Test score: 0.4986230432987213
Test accuracy: 0.8506399989128113
```