200 lines (200 loc) · 12.6 KB

MACHINE LEARNING

SEGUNDA PARTE

M.SC. LUIS FERNANDO ESPINO BARRIOS

2020

3. MÉTODO DE BAYES

CAUSA Y EFECTO

$$P(causa|efecto) = P(efecto|causa) * P(causa) / P(efecto)$$

$$P(efecto|causa) = P(causa|efecto) * P(efecto) / P(causa)$$

$$P(causa|efecto\ 1, \ldots\ efecto\ n) =$$
$$P(causa) * \Pi\ P(efecto/causa)$$

EJEMPLO

Table 1. A small training set

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

- Suponer que N es no_ganar y P es ganar.

- Calcular P(ganar|sunny, hot, high, false) y P(no_ganar|sunny, hot, high, false) para determinar si ese conjunto de efectos son de la clase N o P.

- P(sunny|ganar) = P(ganar|sunny)*P(sunny)/P(ganar)

  2/5*5/14 / 9/14 = 0.4*0.35/0.64 = 0.21

- P(hot|ganar)

  2/4*4/14 / 9/14 = 0.5*0.28/0.64 = 0.21

- P(high|ganar)

  3/7*7/14 / 9/14 = 0.42*0.5/0.64 = 0.32

- P(false|ganar)

  6/8*8/14 / 9/14 = 0.75*0.57/0.64 = 0.66

- P(sunny|ganar) = 0.21
- P(hot|ganar) = 0.21
- P(high|ganar) = 0.32
- P(false|ganar) = 0.66

P(causa|efecto 1, … efecto n) =

P(causa) * Π P(efecto/causa)

0.64 * 0.21 * 0.21 * 0.32 * 0.66 = **0.005**

P(NO JUGAR|SUNNY, HOT, HIGH, FALSE)

- P(sunny|no_ganar) = P(no_ganar|sunny)*P(sunny)/P(no_ganar)

  3/5*5/14 / 5/14 = 0.6*0.35/0.35 = 0.6
- P(hot|no_ganar)

  2/4*4/14 / 5/14 = 0.5*0.28/0.35 = 0.21
- P(high|no_ganar)

  4/7*7/14 / 5/14 = 0.57*0.5/0.35 = 0.81
- P(false|no_ganar)

  2/8*8/14 / 5/14 = 0.25*0.57/0.35 = 0.40

- P(sunny|no_ganar) = 0.6
- P(hot|no_ganar) = 0.21
- P(high|no_ganar) = 0.81
- P(false|no_ganar) = 0.40

$$P(\text{causa}|\text{efecto } 1, \ldots \text{efecto } n) =$$

$$P(\text{causa}) * \Pi\, P(\text{efecto}/\text{causa})$$

$$0.35 * 0.6 * 0.21 * 0.81 * 0.40 = \mathbf{0.014}$$

RESULTADOS

- P(ganar|sunny, hot, high, false) = 0.005
- P(no_ganar|sunny, hot, high, false) = 0.014

Por lo tanto, con esos efectos la clase con mayor probabilidad es no_ganar o N. Lo que corresponde con la tabla.

No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	N

EJEMPLO

Table 1. A small training set

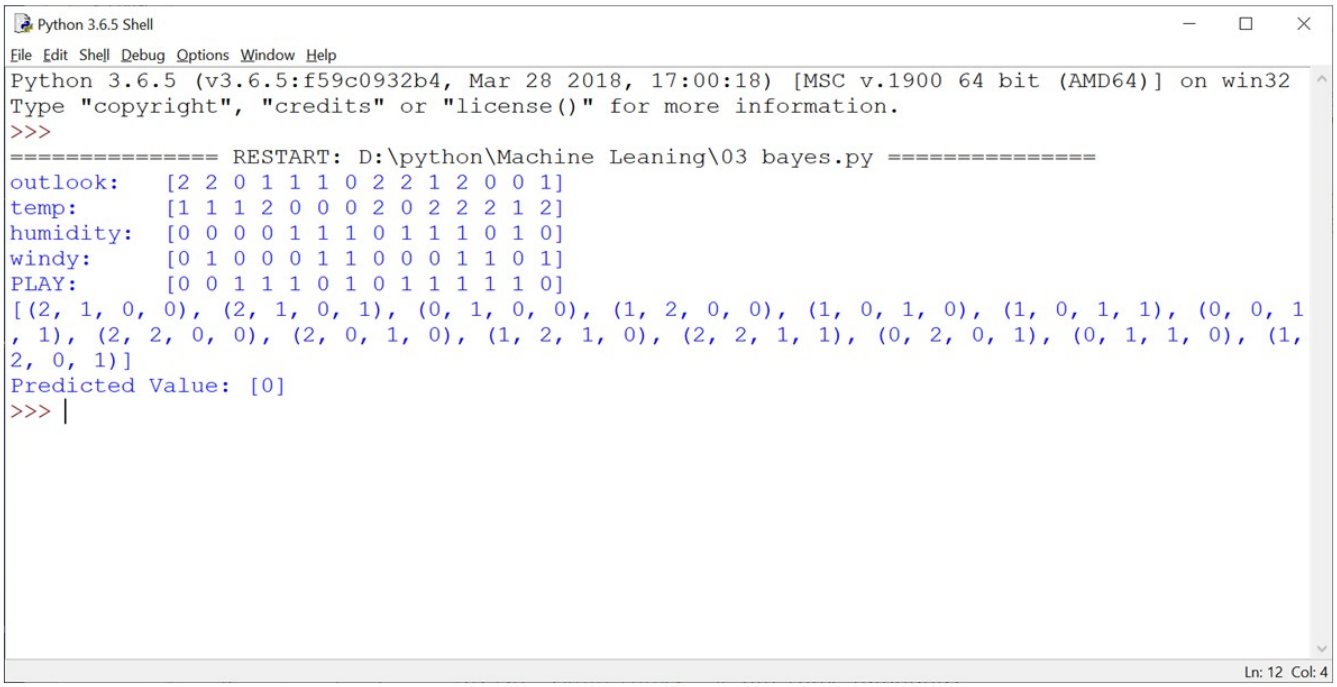
No.	Attributes				Class
	Outlook	Temperature	Humidity	Windy	
1	sunny	hot	high	false	N
2	sunny	hot	high	true	N
3	overcast	hot	high	false	P
4	rain	mild	high	false	P
5	rain	cool	normal	false	P
6	rain	cool	normal	true	N
7	overcast	cool	normal	true	P
8	sunny	mild	high	false	N
9	sunny	cool	normal	false	P
10	rain	mild	normal	false	P
11	sunny	mild	normal	true	P
12	overcast	mild	high	true	P
13	overcast	hot	normal	false	P
14	rain	mild	high	true	N

EJEMPLO CON SCIKIT-LEARN

```
# Data creation
outlook=['sunny','sunny','overcast','rain','rain','rain','overcast',
         'sunny','sunny','rain','sunny','overcast','overcast','rain']
temperature=['hot','hot','hot','mild','cool','cool','cool',
             'mild','cool','mild','mild','mild','hot','mild']
humidity=['high','high','high','high','normal','normal','normal',
          'high','normal','normal','normal','high','normal','high']
windy=['false','true','false','false','false','true','true',
       'false','false','false','true','true','false','true']
play=['N','N','P','P','P','N','P','N','P','P','P','P','P','N']
```

```
# Import LabelEncoder
from sklearn import preprocessing

# Creating labelEncoder
le = preprocessing.LabelEncoder()

# Converting string labels into numbers.
outlook_encoded=le.fit_transform(outlook)
temperature_encoded=le.fit_transform(temperature)
humidity_encoded=le.fit_transform(humidity)
windy_encoded=le.fit_transform(windy)
label=le.fit_transform(play)

print ("outlook:  ",outlook_encoded)
print ("temp:     ",temperature_encoded)
print ("humidity: ",humidity_encoded)
print ("windy:    ",windy_encoded)
print ("PLAY:     ",label)

# Combinig attributes into single listof tuples
features=list(zip(outlook_encoded,temperature_encoded,humidity_encoded,windy_encoded))
print (features)
```

```
# Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

# Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(features,label)

# Predict Output
predicted= model.predict([[2,1,0,0]]) # sunny, hot, high, false
print ("Predicted Value:", predicted)
```



4. REDES NEURONALES

DEFINICIÓN

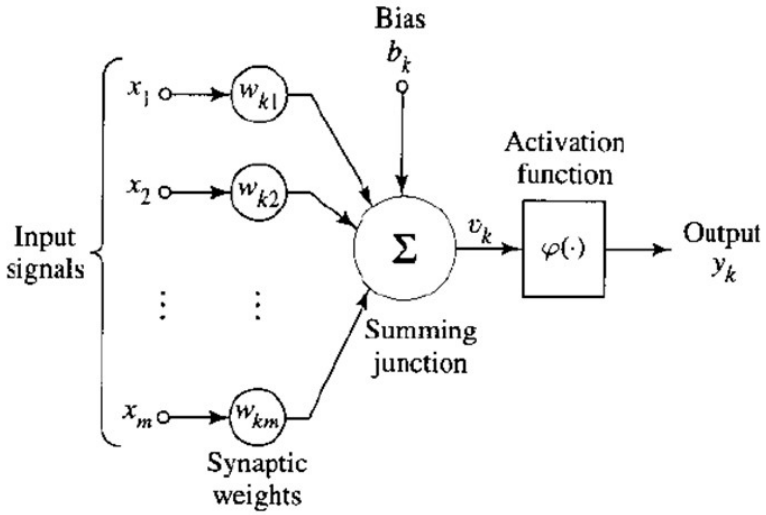
- Es un modelo de aprendizaje basado en el cerebro humano.
- La unidad básica de una red neuronal es una neurona, y sirve para el procesamiento de información en la operación de la red.
- Las redes neuronales están compuestas por unidades que están interconectadas, estos enlaces tienen pesos asociados y existen funciones de activación para que la red alcance un estado estable.

- Las redes neuronales se pueden representar matemáticamente a través de grafos dirigidos.
- Estas redes se basan en las redes de Hopfield, y en los preceptrons, según su arquitectura se clasifican en redes de capa simple, multicapa y redes recurrentes, la diferencia de estas últimas es la presencia de ciclos.

- Existen muchas aplicaciones donde se utilizan las redes neuronales, entre las cuales se pueden mencionar el reconocimiento de voz, clasificación, agrupación, predicción, identificación de patrones y análisis de datos.

PERCEPTRON

- Un perceptron es una máquina o sistema nervioso hipotético desde la perspectiva empirista. Es diseñado para ilustrar algunas propiedades fundamentales de los sistemas inteligentes en general, haciendo evidente la analogía con los sistemas biológicos.



FUNCIÓN SIGMOIDE

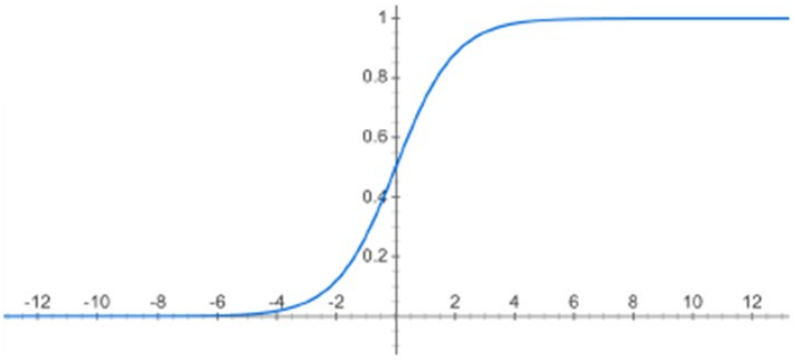


Figure 1: $\frac{1}{1+e^{-x}}$.

EJEMPLO 1

- Si se desea simular una compuerta OR con un perceptrón se debe agregar una variable de tipo bias o threshold, que sirve para limitar o establecer una tendencia.

$$y = \mathrm{w0} + x1 * w1 + x2 * w2$$

Posible solución

- W0 = -0.5
- W1 = 1
- W2 = 1

EJEMPLO 2

- Si se desea simular una compuerta AND con un perceptrón se debe agregar una variable de tipo bias o threshold, que sirve para limitar o establecer una tendencia.

$$y = \mathrm{w0} + x1 * w1 + x2 * w2$$

Posible solución

- W0 = -1.5
- W1 = 1
- W2 = 1

EJEMPLO 3

- Si se desea simular una compuerta XOR con un perceptrón se debe agregar una variable de tipo bias o threshold, que sirve para limitar o establecer una tendencia.

$$y = \mathrm{w0} + x1 * w1 + x2 * w2$$

Posible solución: multicapa

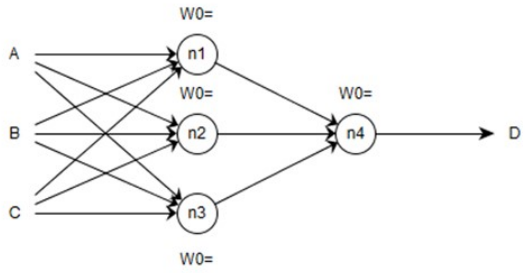
- Neurona 1: W0 = -0.5; W1 = 1; W2 = -1
- Neurona 2: W0 = -0.5; W1 = -1; W2 = 1
- Neurona 3: W0 = -0.5; W1 = 1; W2 = 1

EJEMPLO 4

- Considerar la expresión (A OR B) XOR C, la tabla de verdad sería de esta forma:

A	B	C	(AVB)XOR C
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

A continuación se presenta una manera de entrenamiento de una red neuronal para procesar mediante tres entradas A, B, C valores lógicos mediante 8 símbolos. La red neuronal utiliza el siguiente diagrama:



	w0	w1	w2	w3
N1	-0.5	-1	-1	1
N2	-0.5	-1	1	-1
N3	-0.5	1	0	-1
N4	-0.5	1	1	1

Con estos valores, la red neuronal proporcionará el siguiente comportamiento:

A	B	C	(AVB)XOR C	n1	n2	n3	n4
0	0	0	**0**	0	0	0	**0**
0	0	1	**1**	1	0	0	**1**
0	1	0	**1**	0	1	0	**1**
0	1	1	**0**	0	0	0	**0**
1	0	0	**1**	0	0	1	**1**
1	0	1	**0**	0	0	0	**0**
1	1	0	**1**	0	0	1	**1**
1	1	1	**0**	0	0	0	**0**

EJEMPLO DE XOR EN SCIKIT

```
# Import Perceptron
from sklearn.neural_network import MLPClassifier

# Training set
x=[[0,0],[0,1],[1,0],[1,1]]
y = [0,1,1,0]

# Create model
model = MLPClassifier(activation='logistic', max_iter=200,
                      hidden_layer_sizes=(4,), alpha=0.001,
                      solver='lbfgs')

# Train the model
model.fit(x,y)

# Predict Output
print('predictions:', model.predict(x))
```

Parameters:

hidden_layer_sizes : *tuple, length = n_layers - 2, default=(100,)*
The ith element represents the number of neurons in the ith hidden layer.

activation : *{'identity', 'logistic', 'tanh', 'relu'}, default='relu'*
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max(0, x)

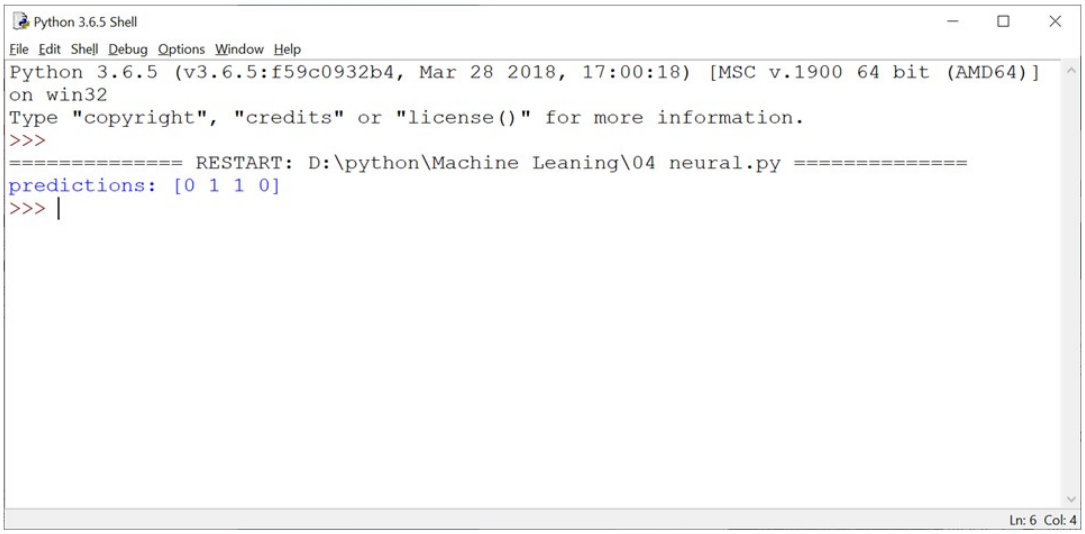
solver : *{'lbfgs', 'sgd', 'adam'}, default='adam'*
The solver for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

Note: The default solver 'adam' works pretty well on relatively large datasets (with thousands of training samples or more) in terms of both training time and validation score. For small datasets, however, 'lbfgs' can converge faster and perform better.

alpha : *float, default=0.0001*
L2 penalty (regularization term) parameter.

max_iter : *int, default=200*
Maximum number of iterations. The solver iterates until convergence (determined by 'tol') or this number of iterations. For stochastic solvers ('sgd', 'adam'), note that this determines the number of epochs (how many times each data point will be used), not the number of gradient steps.



HOJA DE TRABAJO: SCIKIT

Utilizar el ejemplo 4 de redes neuronales

1. Hacer la predicción de todos los valores con Bayes
2. Hacer la predicción de todos los valores con Redes Neuronales

TAREA 8 (FIG. 18.3 NORVIG)

- Mediante Scikit-learn predecir todos los valores con Bayes y con Redes Neuronales:

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*60*	$_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*60*	$_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$_{12} = Yes$

Figure 18.3 Examples for the restaurant domain.

REFERENCIAS

- Quinlan, J. R. (1986). Induction of decision Trees. *Machine Learning, 1*(1), 81-106.
- Nilsson, N. (1998). *Artificial Intelligence: A New Synthesis.* United States of America: Morgan Kaufmaann Publisher, Inc.
- Russell, S., & Norvig, P. (2003). *Artificial Intelligence: A Modern Approach* (2 ed.). United States of America: Pearson Education, Inc.
- Guía de Usuario de scikit-learn https://scikit-learn.org/stable/user_guide.html