Recomendación de campañas

November 9, 2019

1 Modelo de cascada

Autor: Andres Felipe Camargo Lastra

1.1 Definición de requisitos

1. Desarrollar un algoritmo que permita predecir la recomendación de campañas para el uso de puntos de tarjeta de credito, basado en un clasificador bayesiano.
2. Desarrollar pruebas unitarias para validar las predicciones del algoritmo.

1.2 Diseño del software y del sistema

Se desea desarrollar un algoritmo para la recomendación de campañas para el uso de puntos de tarjetas de crédito. Para poder realizar el algoritmo hay que basarse en el teorema de Bayes; por tal razon se va utilizar un clasificador bayesiano.

Se debe leer un dataset del banco de la alegría utilizando la librería pandas y trabajar con las primeras 7 filas. Se le asignará un valor unico númerico a cada elemento de las columnas. Las columnas del dataset son (Tipo de ocupación, Mayor a 30, Volumen de compras, Mayoria de elementos comprados, Campaña), cabe aclarar que lo que se quiere predecir es la columna Campaña.

A continuación se muestran los valores de las columnas con sus respectivos valores númericos:

Tipo de ocupación

Empleado: 0

Independiente: 1

Mayor a 30

Si: 1

No: 0

Mayor a 30

Si: 1

No: 0

Volumen de compras

Alto: 0

Medio: 2

Bajo: 1

Mayoría de elementos comprados

Ferretería: 0

Vestuario: 2

Mercado: 1

Campaña

Salud: 1

Hogar: 0

Viajes: 2

1.3 Implementación y pruebas unitarias

Importamos la librería Pandas para leer el conjunto de datos del banco de la alegría.

[43]:
```
import pandas as pd
```

Almacenamos el conjunto de datos en una variable y mostramos las primeras 7 filas.

[44]:
```
Datos = pd.read_csv('Datos_Balegria.csv')
Datos.head(7)
#Datos.Campaña.unique()
```

[44]:

	Tipo de Ocupación	Mayor a 30	Volumen de Compras	Mayoria de elementos comprados	Campaña
0	Empleado	Si	Alto	Ferretería	Salud
1	Independiente	Si	Medio	Vesturario	Hogar
2	Independiente	No	Alto	Vesturario	Viajes
3	Independiente	Si	Alto	Mercado	Hogar
4	Empleado	Si	Bajo	Mercado	Viajes
5	Empleado	Si	Medio	Vesturario	Salud
6	Empleado	No	Alto	Vesturario	Viajes

Ahora se escogen los datos para realizar el entrenamiento de la predicción.

[45]:
```
features_train = Datos.iloc[0:7,0:4]
target_train = Datos.iloc[0:7,4]
```

El siguiente paso es importar la librería preprocessing de sklearn, esta permite asignarle un valor unico númerico a cada elemento.

[46]:
```
#import LabelEncoder
from sklearn import preprocessing
```

Se procede a realizar la transformación de valores en números para cada columna y se muestra al final los valores de las mismas.

[47]:
```
le = preprocessing.LabelEncoder()
f0 = le.fit_transform(features_train.iloc[0:7,0])
f1 = le.fit_transform(features_train.iloc[0:7,1])
f2 = le.fit_transform(features_train.iloc[0:7,2])
f3 = le.fit_transform(features_train.iloc[0:7,3])
label = le.fit_transform(target_train)
features = list(zip(f0,f1,f2,f3))
print(features)
print(label)
```

```
[(0, 1, 0, 0), (1, 1, 2, 2), (1, 0, 0, 2), (1, 1, 0, 1), (0, 1, 1, 1), (0, 1, 2,
2), (0, 0, 0, 2)]
[1 0 2 0 2 1 2]
```

Por ultimo utilizamos un modelo Gaussiano para predecir cual es la campaña que se le va ofrecer al usuario.

[48]:
```
from sklearn.naive_bayes import GaussianNB
model1 = GaussianNB()
model1.fit(features, label)
Predicted = model1.predict([[1,1,0,2]])
#Predicted = model1.predict([[0,1,2,1]])
#Predicted = model1.predict([[0,0,2,1]])
Predicted = model1.predict([[0, 1, 0, 0]])
print(Predicted)
```

```
[1]
```

1.3.1 Pruebas Unitarias

Para realizar pruebas unitarias haremos uso de la libreria unittest. Los datos a Probar son los siguientes: Tipo de Ocupación Mayor a 30 Volumen de Compras Mayoria de elementos comprados Campaña Empleado Si Medio Mercado Hogar Independiente Si Alto Vesturario Viajes En este caso observamos que ninguna de las dos pruebas cumple con las predicciones que se realizaron.

[49]:
```
import unittest
from sklearn.naive_bayes import GaussianNB

def bayesiano(a,b,c,d):
    from sklearn.naive_bayes import GaussianNB
    model1 = GaussianNB()
    model1.fit(features, label)
    Predicted = model1.predict([[a,b,c,d]])
    return Predicted

class TestStringMethods(unittest.TestCase):

    def test1_(self):
        predict = bayesiano(0,1,2,1)
        self.assertEqual(0, predict[0])
    def test2_(self):
        predict = bayesiano(1, 1, 0, 2)
        self.assertEqual(2, predict[0])

unittest.main(argv=['ignored', '-v'], exit=False)
```

```
test1_ (__main__.TestStringMethods) … FAIL
test2_ (__main__.TestStringMethods) … FAIL

======================================================================
FAIL: test1_ (__main__.TestStringMethods)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-49-ca64c4f1ba9c>", line 15, in test1_
    self.assertEqual(0, predict[0])
AssertionError: 0 != 1

======================================================================
FAIL: test2_ (__main__.TestStringMethods)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "<ipython-input-49-ca64c4f1ba9c>", line 18, in test2_
    self.assertEqual(2, predict[0])
AssertionError: 2 != 0

----------------------------------------------------------------------
Ran 2 tests in 0.015s

FAILED (failures=2)
```

[49]:
```
<unittest.main.TestProgram at 0x7f2a45d7e438>
```

1.4 Integración y pruebas del sistema

Esta aplicación podría ser integrada con una base datos en MongoDB en la cual se puedan tener millones de registros y luego realizar pruebas para predecir la campaña que se le quiere ofrecer al usuario.

Para lograr la integración con MongoDB haremos uso de la librería PyMongo, la cual permite realizar la conexión a este motor de base de datos.

1.5 Operación y mantenimiento

Este aplicativo se pondrá en marcha en el banco de la alegría y se le ofrece mantenimiento las 24 horas del día. A medida que el aplicativo este escalando se hace necesario utilizar un servicio cloud como AWS para el almacenamiento y su seguridad.