Proyecto_seminario

July 8, 2019

1 Aplicación de un modelo oculto de Márkov para determinar el grado de honestidad de una persona al responder preguntas.

1.1 Dependencias:

Para la realización de este proyecto se necesitaron las siguientes librerías:

1. numpy: Es un paquete fundamental necesario para la computación científica con Python.
2. pandas: Es una biblioteca de código abierto que proporciona estructuras de datos de alto rendimiento y fáciles de usar, y herramientas de análisis de datos.
3. NetworkX: Es un paquete de Python para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas.
4. Matplotlib: Es una biblioteca para hacer diagramas 2D de matrices en Python.

1.2 Descripción del modelo:

Este modelo permite descubrir si una persona es honesta por medio de "señales honestas" luego de realizarle varias preguntas en una entrevista.

El modelo está compuesto de tres estados ocultos(deshonesto, insierto y honesto) con probabilidades iniciales de 1/3 cada uno, cinco estados observables(cubrirse la boca, mantener una mirada fija, mantener un tono de voz inestable, tocarse la nariz, rascarse el cuello o cubrir el área de la garganta), la matriz de transición de estados (MxM), la matriz de emisión (MxO) y una secuencia de datos de observación.

La idea es determinar cual es el estado oculto mas probable al responder una pregunta.

1.3 Implementación del modelo:

[1]:
```
import numpy as np
import pandas as pd
import networkx as nx
from matplotlib import pyplot as plt
%matplotlib inline
from pprint import pprint
```

[2]:
```
def viterbi(pi, a, b, obs):

    nStates = np.shape(b)[0]
    T = np.shape(obs)[0]

    # Camino inicial. Matrices con ceros.
    path = np.zeros(T)
    # delta --> mayor probabilidad de cualquier camino que alcance el estado i.
    delta = np.zeros((nStates, T))
    # phi --> argmax por paso de tiempo para cada estado
    phi = np.zeros((nStates, T))

    # inicio
    delta[:, 0] = pi * b[:, obs[0]]
    phi[:, 0] = 0

    print('\nInicio. Caminar hacia adelante\n')
    # Extensión del algoritmo directo
    for t in range(1, T):
        for s in range(nStates):

            delta[s, t] = np.max(delta[:, t-1] * a[:, s]) * b[s, obs[t]]
            phi[s, t] = np.argmax(delta[:, t-1] * a[:, s])
            print('s={s} and t={t}: phi[{s}, {t}] = {phi}'.format(s=s, t=t,␣
↪phi=phi[s, t]))

    # Encontrar el camino óptimo
    print('-'*50)
    print('Iniciar retroceso\n')
    path[T-1] = np.argmax(delta[:, T-1])


    for t in range(T-2, -1, -1):

        x = int(path[t+1])

        path[t] = phi[x, [t+1]]
        print('path[{}] = {}'.format(t, path[t]))

    return path, delta, phi
```

1.4 Descripción del entrenamiento:

Se utilizó el algoritmo de Baum-Welch para definir la matriz de transición de estados ocultos y la matriz de emision u observación. Este algoritmo de entrenamiento necesita como parámetros los estados observables, los estados ocultos y la probabilidad inicial de los estados ocultos. Luego estas matrices se pasan como parámetros al algoritmo de Viterbi junto con la probabilidad inicial

de los estados ocultos para conseguir la secuencia mas probable de estados ocultos (deshonesto, neutral y honesto) que es producida por una secuencia de estados observables(cubrirse la boca, mantener una mirada fija, mantener un tono de voz inestable, tocarse la nariz y cubrir el área de la garganta) con el fin de determinar si la persona entrevistada esta siendo honesta con sus respuestas.

[3]:
```
def forward(V, a, b, initial_distribution):
    alpha = np.zeros((V.shape[0], a.shape[0]))
    alpha[0, :] = initial_distribution * b[:, V[0]]

    for t in range(1, V.shape[0]):
        for j in range(a.shape[0]):
            # Matrix Computation Steps
            #                  ((1x2) . (1x2))      *     (1)
            #                        (1)            *     (1)
            alpha[t, j] = alpha[t - 1].dot(a[:, j]) * b[j, V[t]]

    return alpha


def backward(V, a, b):
    beta = np.zeros((V.shape[0], a.shape[0]))

    # setting beta(T) = 1
    beta[V.shape[0] - 1] = np.ones((a.shape[0]))

    # Loop in backward way from T-1 to
    # Due to python indexing the actual loop will be T-2 to 0
    for t in range(V.shape[0] - 2, -1, -1):
        for j in range(a.shape[0]):
            beta[t, j] = (beta[t + 1] * b[:, V[t + 1]]).dot(a[j, :])
    return beta


def baum_welch(V, a, b, initial_distribution, n_iter=100):
    M = a.shape[0]
    T = len(V)

    for n in range(n_iter):
        alpha = forward(V, a, b, initial_distribution)
        beta = backward(V, a, b)

        xi = np.zeros((M, M, T - 1))
        for t in range(T - 1):
            denominator = np.dot(np.dot(alpha[t, :].T, a) * b[:, V[t + 1]].T,␣
 ↪beta[t + 1, :])
            for i in range(M):
```

```
                numerator = alpha[t, i] * a[i, :] * b[:, V[t + 1]].T * beta[t +␣
↪1, :].T
                xi[i, :, t] = numerator / denominator

        gamma = np.sum(xi, axis=1)
        a = np.sum(xi, 2) / np.sum(gamma, axis=1).reshape((-1, 1))

        # Add additional T'th element in gamma
        gamma = np.hstack((gamma, np.sum(xi[:, :, T - 2], axis=0).reshape((-1,␣
↪1))))

        K = b.shape[1]
        denominator = np.sum(gamma, axis=1)
        for l in range(K):
            b[:, l] = np.sum(gamma[:, V == l], axis=1)

        b = np.divide(b, denominator.reshape((-1, 1)))


    return a, b
```

Se pasan los parametros al modelo de entrenamiento:

[4]:
```
data = pd.read_csv('/home/katherine/compu_kathe/9no_semestre/seminario/
↪proyecto_seminario/prueba1.csv')

V = data['visible'].values
obs_map = {'CB':0, 'TN':1, 'CG':2, 'MF':3, 'VI':4, 'NO':5}
#CB = cubrir boca, MF = Mirada fija, VI= tono voz inestable, CG = cubrirse␣
↪garganta TN = tocarse la nariz,
#NO = niguna de las anteriores.

array = []

for i in V:
    #print('\n i: ',i)
    for k, v in obs_map.items():
        # print('k : ',k,' v: ', v)
        if i == k:
            array.append(v)

obs = np.array(array)

# Transition Probabilities
a = np.ones((3, 3))
a = a / np.sum(a, axis=1)

#Estas son las probabilidades de estado inicial
```

```
pi = a[0]

# Emission Probabilities
b = np.array(((1, 3, 5, 1, 3, 5), (2, 4, 6, 2, 4, 6),(1, 3, 5, 1, 3, 5)))
#print('B: ',b)
b = b / np.sum(b, axis=1).reshape((-1, 1))

a, b= baum_welch(obs, a, b, pi, n_iter=100)

print('MATRIZ A: ', a, '\n')
print('MATRIZ B: ', b, '\n')
```

```
MATRIZ A:  [[2.09834444e-01 5.80331112e-01 2.09834444e-01]
 [2.65586933e-05 9.99946883e-01 2.65586933e-05]
 [2.09834444e-01 5.80331112e-01 2.09834444e-01]]

MATRIZ B:  [[9.56372350e-132 1.00000000e+000 1.28337566e-010 4.23819947e-131
  1.41836799e-132 8.10545154e-094]
 [1.19903231e-001 1.60677384e-001 2.39806462e-001 1.19903231e-001
  1.19903231e-001 2.39806462e-001]
 [9.56372350e-132 1.00000000e+000 1.28337566e-010 4.23819947e-131
  1.41836799e-132 8.10545154e-094]]
```

Se muestran las matrices de transición y observación

[5]:
```
#Se asigna a un arreglo los estados ocultos
print('Probabilidades Iniciales de los Estados Ocultos: \n')
hidden_states = ['deshonesto', 'incierto', 'honesto']

state_space = pd.Series(pi, index=hidden_states, name='Estados')
print(state_space)
print('\n', state_space.sum())
print('\n')

#Matriz de probabilidad de transición de estados cambiantes dado un estado
#Se crea una matriz (MxM) donde M es el número de estados.

print('Matriz de Transición De modelo: \n')
a_df = pd.DataFrame(columns=hidden_states, index=hidden_states)
a_df.loc[hidden_states[0]] = a[0]
a_df.loc[hidden_states[1]] = a[1]
a_df.loc[hidden_states[2]] = a[2]

print(a_df)

a_model = a_df.values
print('\n', a_model, a_model.shape, '\n')
```

```
print(a_df.sum(axis=1))
```

```
Probabilidades Iniciales de los Estados Ocultos:

deshonesto    0.333333
incierto      0.333333
honesto       0.333333
Name: Estados, dtype: float64

 1.0


Matriz de Transición De modelo:

             deshonesto  incierto      honesto
deshonesto     0.209834  0.580331     0.209834
incierto    2.65587e-05  0.999947  2.65587e-05
honesto        0.209834  0.580331     0.209834

 [[0.20983444405145937 0.580331111897081 0.20983444405145937]
 [2.655869325990881e-05 0.9999468826134802 2.655869325990881e-05]
 [0.20983444405145937 0.580331111897081 0.20983444405145937]] (3, 3)

deshonesto    1.0
incierto      1.0
honesto       1.0
dtype: float64
```

[6]:
```
#Matriz de probabilidad de emisión u observación.
#b = probabilidad de observación dada el estado.
#La matriz es el tamaño (M x O) donde M es el número de estados
#y O es el número de diferentes observaciones posibles.

print('\n Matriz de probabilidad de Emisión u observación del entrenamiento: \n')
states = ['CB', 'TN', 'CG', 'MF', 'VI', 'NO']
observable_states = states

b_df = pd.DataFrame(columns=observable_states, index=hidden_states)
b_df.loc[hidden_states[0]] = b[0]
b_df.loc[hidden_states[1]] = b[1]
b_df.loc[hidden_states[2]] = b[2]

print(b_df)

b_model = b_df.values
print('\n', b_model, b_model.shape, '\n')
print(b_df.sum(axis=1))
```

```
print('\n')
```

```
 Matriz de probabilidad de Emisión u observación del entrenamiento:

                      CB        TN           CG           MF            VI  \
deshonesto  9.56372e-132         1  1.28338e-10  4.2382e-131  1.41837e-132
incierto        0.119903  0.160677     0.239806     0.119903      0.119903
honesto     9.56372e-132         1  1.28338e-10  4.2382e-131  1.41837e-132

                     NO
deshonesto  8.10545e-94
incierto       0.239806
honesto     8.10545e-94

 [[9.563723497050358e-132 0.9999999998716623 1.2833756645962422e-10
  4.23819947420096e-131 1.4183679870705015e-132 8.10545153668598e-94]
 [0.11990323086750634 0.16067738395299896 0.23980646170946943
  0.11990323086750634 0.11990323086750634 0.2398064617350127]
 [9.563723497050358e-132 0.9999999998716623 1.2833756645962422e-10
  4.23819947420096e-131 1.4183679870705015e-132 8.10545153668598e-94]] (3, 6)

deshonesto    1.0
incierto      1.0
honesto       1.0
dtype: float64
```

Se inserta la secuencia de observaciones del entrevistado:

[7]:
```
#Secuencia de Observaciones.
obs_vi = np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1])

array_nstates = ([])
inv_obs_map = dict((v,k) for k, v in obs_map.items())

obs_seq = [inv_obs_map[v] for v in list(obs_vi)]

print( pd.DataFrame(np.column_stack([obs_vi, obs_seq]),
                columns=['Código', 'Secuencia']) )
```

```
   Código Secuencia
0       0        CB
1       0        CB
2       0        CB
3       0        CB
4       0        CB
5       0        CB
6       0        CB
7       0        CB
8       0        CB
9       0        CB
10      0        CB
11      0        CB
12      0        CB
13      0        CB
14      0        CB
15      0        CB
16      0        CB
17      0        CB
18      1        TN
19      1        TN
20      1        TN
21      1        TN
```

Se llama la función de Viterbi para calcular el estado oculto mas probable:

[8]:
```
path, delta, phi = viterbi(pi, a_model, b_model, obs_vi)
print('\n La mejor ruta del estado: \n', path)
print('\n delta:\n', delta)
print('phi:\n', phi)
```

```
Inicio. Caminar hacia adelante

s=0 and t=1: phi[0, 1] = 1.0
s=1 and t=1: phi[1, 1] = 1.0
s=2 and t=1: phi[2, 1] = 1.0
s=0 and t=2: phi[0, 2] = 1.0
s=1 and t=2: phi[1, 2] = 1.0
s=2 and t=2: phi[2, 2] = 1.0
s=0 and t=3: phi[0, 3] = 1.0
s=1 and t=3: phi[1, 3] = 1.0
s=2 and t=3: phi[2, 3] = 1.0
s=0 and t=4: phi[0, 4] = 1.0
s=1 and t=4: phi[1, 4] = 1.0
s=2 and t=4: phi[2, 4] = 1.0
s=0 and t=5: phi[0, 5] = 1.0
s=1 and t=5: phi[1, 5] = 1.0
s=2 and t=5: phi[2, 5] = 1.0
```

```
s=0 and t=6: phi[0, 6] = 1.0
s=1 and t=6: phi[1, 6] = 1.0
s=2 and t=6: phi[2, 6] = 1.0
s=0 and t=7: phi[0, 7] = 1.0
s=1 and t=7: phi[1, 7] = 1.0
s=2 and t=7: phi[2, 7] = 1.0
s=0 and t=8: phi[0, 8] = 1.0
s=1 and t=8: phi[1, 8] = 1.0
s=2 and t=8: phi[2, 8] = 1.0
s=0 and t=9: phi[0, 9] = 1.0
s=1 and t=9: phi[1, 9] = 1.0
s=2 and t=9: phi[2, 9] = 1.0
s=0 and t=10: phi[0, 10] = 1.0
s=1 and t=10: phi[1, 10] = 1.0
s=2 and t=10: phi[2, 10] = 1.0
s=0 and t=11: phi[0, 11] = 1.0
s=1 and t=11: phi[1, 11] = 1.0
s=2 and t=11: phi[2, 11] = 1.0
s=0 and t=12: phi[0, 12] = 1.0
s=1 and t=12: phi[1, 12] = 1.0
s=2 and t=12: phi[2, 12] = 1.0
s=0 and t=13: phi[0, 13] = 1.0
s=1 and t=13: phi[1, 13] = 1.0
s=2 and t=13: phi[2, 13] = 1.0
s=0 and t=14: phi[0, 14] = 1.0
s=1 and t=14: phi[1, 14] = 1.0
s=2 and t=14: phi[2, 14] = 1.0
s=0 and t=15: phi[0, 15] = 1.0
s=1 and t=15: phi[1, 15] = 1.0
s=2 and t=15: phi[2, 15] = 1.0
s=0 and t=16: phi[0, 16] = 1.0
s=1 and t=16: phi[1, 16] = 1.0
s=2 and t=16: phi[2, 16] = 1.0
s=0 and t=17: phi[0, 17] = 1.0
s=1 and t=17: phi[1, 17] = 1.0
s=2 and t=17: phi[2, 17] = 1.0
s=0 and t=18: phi[0, 18] = 1.0
s=1 and t=18: phi[1, 18] = 1.0
s=2 and t=18: phi[2, 18] = 1.0
s=0 and t=19: phi[0, 19] = 0.0
s=1 and t=19: phi[1, 19] = 1.0
s=2 and t=19: phi[2, 19] = 0.0
s=0 and t=20: phi[0, 20] = 0.0
s=1 and t=20: phi[1, 20] = 1.0
s=2 and t=20: phi[2, 20] = 0.0
s=0 and t=21: phi[0, 21] = 0.0
s=1 and t=21: phi[1, 21] = 1.0
s=2 and t=21: phi[2, 21] = 0.0
```

```
-------------------------------------------------
Iniciar retroceso

path[20] = 1.0
path[19] = 1.0
path[18] = 1.0
path[17] = 1.0
path[16] = 1.0
path[15] = 1.0
path[14] = 1.0
path[13] = 1.0
path[12] = 1.0
path[11] = 1.0
path[10] = 1.0
path[9] = 1.0
path[8] = 1.0
path[7] = 1.0
path[6] = 1.0
path[5] = 1.0
path[4] = 1.0
path[3] = 1.0
path[2] = 1.0
path[1] = 1.0
path[0] = 1.0

 La mejor ruta del estado:
 [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

 delta:
 [[3.18790783e-132 1.01518068e-137 1.21716978e-138 1.45934837e-139
  1.74971290e-140 2.09785086e-141 2.51525735e-142 3.01571464e-143
  3.61574721e-144 4.33516744e-145 5.19772972e-146 6.23191483e-147
  7.47187032e-148 8.95853804e-149 1.07410060e-149 1.28781291e-150
  1.54404727e-151 1.85126422e-152 2.32086144e-022 4.86996669e-023
  1.02188675e-023 2.14427039e-024]
 [3.99677436e-002 4.79200704e-003 5.74546606e-004 6.88863351e-005
  8.25925541e-006 9.90258805e-007 1.18728923e-007 1.42352253e-008
  1.70675884e-009 2.04635029e-010 2.45350979e-011 2.94168124e-012
  3.52698350e-013 4.22874253e-014 5.07012960e-015 6.07892628e-016
  7.28844185e-017 8.73861306e-018 1.40402290e-018 2.25582744e-019
  3.62441199e-020 5.82330104e-021]
 [3.18790783e-132 1.01518068e-137 1.21716978e-138 1.45934837e-139
  1.74971290e-140 2.09785086e-141 2.51525735e-142 3.01571464e-143
  3.61574721e-144 4.33516744e-145 5.19772972e-146 6.23191483e-147
  7.47187032e-148 8.95853804e-149 1.07410060e-149 1.28781291e-150
  1.54404727e-151 1.85126422e-152 2.32086144e-022 4.86996669e-023
  1.02188675e-023 2.14427039e-024]]
phi:
```

```
[[0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
 [0. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0.]]
```

1.5 Descripción de las pruebas:

En este experimento se entrevistaron a 20 personas, a cada una de ellas se les hizo 40 preguntas por lo que se obtuvieron 800 registros en total. Estas respuestas permiten observar si una persona es honesta, deshonesta o insierto con respecto a los estados observables.

Estos registros se introducen en archivos txt para pasarselos como parámetros al algoritmo de entrenamiento Baum-Welch.

1.6 Implementación de las pruebas:

[ ]: