Instituto Politécnico Nacional

Escuela Superior de Cómputo

Aprendizaje no supervisado, Kmeans

Natural Language Processing

Estudiante:
Nicolás Sayago Abigail

Profesora:
Olga Kolesnicova

Mayo 11, 2020

1 Kmeans

ITERACIONES

Algunos de los resultados obtenidos con 100 iteraciones. Para generar estos resultados tuve que balancear el número de mensajes HAM y SPAM en el corpus. Cuando habían más mensajes HAM, todos los datos se iban a un solo cluster.

```
---------------------------------
Cluster 1: HAM: 25  SPAM: 1
Cluster 2: HAM: 400  SPAM: 322
---------------------------------
Cluster 1: HAM: 185  SPAM: 292
Cluster 2: HAM: 240  SPAM: 31
---------------------------------
Cluster 1: HAM: 7  SPAM: 15
Cluster 2: HAM: 418  SPAM: 308
---------------------------------
Cluster 1: HAM: 321  SPAM: 7
Cluster 2: HAM: 104  SPAM: 316
---------------------------------
Cluster 1: HAM: 424  SPAM: 314
Cluster 2: HAM: 1  SPAM: 9
---------------------------------
Cluster 1: HAM: 236  SPAM: 292
Cluster 2: HAM: 189  SPAM: 31
---------------------------------
Cluster 1: HAM: 212  SPAM: 301
Cluster 2: HAM: 213  SPAM: 22
---------------------------------
```

```
---------------------------------
Cluster 1: HAM: 271  SPAM: 255
Cluster 2: HAM: 154  SPAM: 68
---------------------------------
Cluster 1: HAM: 366  SPAM: 112
Cluster 2: HAM: 59  SPAM: 211
---------------------------------
Cluster 1: HAM: 300  SPAM: 126
Cluster 2: HAM: 125  SPAM: 197
---------------------------------
Cluster 1: HAM: 209  SPAM: 323
Cluster 2: HAM: 216  SPAM: 0
---------------------------------
Cluster 1: HAM: 406  SPAM: 322
Cluster 2: HAM: 19  SPAM: 1
---------------------------------
```

Después de esas pruebas, hice otras imprimiendo el valor de la función **Distortion**.

```
---------------------------------
Cluster 1: HAM: 161  SPAM: 227
Cluster 2: HAM: 264  SPAM: 96
Distortion: 0.8517811797636654
---------------------------------
Cluster 1: HAM: 22  SPAM: 164
Cluster 2: HAM: 403  SPAM: 159
Distortion: 0.7994968890407222
---------------------------------
Cluster 1: HAM: 366  SPAM: 29
Cluster 2: HAM: 59  SPAM: 294
Distortion: 0.8509550685981405
---------------------------------
Distortion: 0.6838789412919748
---------------------------------
Distortion: 0.6782177490014893
---------------------------------
Cluster 1: HAM: 168  SPAM: 198
Cluster 2: HAM: 257  SPAM: 125
Distortion: 0.8519123722312913
---------------------------------
```

```
---------------------------------
Cluster 1: HAM: 365  SPAM: 66
Cluster 2: HAM: 60  SPAM: 257
Distortion: 0.8468399670836602
---------------------------------
Distortion: 0.6855458323714845
---------------------------------
Cluster 1: HAM: 234  SPAM: 100
Cluster 2: HAM: 191  SPAM: 223
Distortion: 0.8499741662666599
---------------------------------
```

2 Conclusión

En mi opinión, a pesar de que el algoritmo supervisado era más complejo, considero que se tenían mejores resultados.

✓Código fuente

```
import nltk
import re
import math
import random
from bs4 import BeautifulSoup
from pickle import dump, load
from nltk.corpus import cess_esp
from nltk.corpus import PlaintextCorpusReader
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import numpy as np

##############################################
#                            NORMALIZATION
##############################################
#Parameters: File path, encoding
#Return: String with only lower case letters
#Notes: path =
 ↪  '/Users/27AGO2019/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/corpus/e961024
def getText(corpusRoot, code):
        f = open(corpusRoot, encoding = code) #Cod: utf-8, latin-1
        text = f.read()
        f.close()
        soup = BeautifulSoup(text, 'lxml')
        text = soup.get_text()
        text = text.lower()
        return text

#Parameters: Text
#Return: List of original tokens
def getTokens(text):
        tokens = nltk.word_tokenize(text)
        return tokens

def getWords(fpath, code):
        f = open(fpath, encoding = code) #Cod: utf-8, latin-1
        text = f.read()
```

```
        f.close()

        words = re.sub(" ", " ",  text).split()
        return words

##############################################
#                                    LEMMAS
##############################################
# Return: Dictionary
def createDicLemmas(tokensLemmas):
        lemmas = {}
        j = 0
        for i in range(0, len(tokensLemmas)- 2, 3):
                word = tokensLemmas[i]
                tag = tokensLemmas[i+1]
                val = tokensLemmas[i+2]
                l = (word, tag[0].lower())
                lemmas[l] = val
                j = j+1
        return lemmas

##############################################
#                                FRECUENCY
##############################################
def getVectors(vocabulary, matrix):
        vectors = []
        for x in matrix:
                vector = []
                for word in vocabulary:
                        frec = x.count(word)
                        vector.append(frec)
                vectors.append(vector)
        return vectors

def getFrecuency(vectors):
        matrix = []
        for vector in vectors:
                aux = np.array(vector)
                total = np.sum(aux)
                p = []
                p.append(1)
                for element in vector:
                        ans = element / total
                        p.append(ans)
```

```
                auxNP = np.array(p)
                matrix.append(auxNP)
        m = np.array(matrix)
        return m

###############################################
#                                     TAGGING
###############################################
def tag(tokens):
        s_tagged = nltk.pos_tag(tokens)
        l = list()
        for tag in s_tagged:
                pos = 'n'
                if len(tag[1][0]) > 0:
                        pos = tag[1][0].lower()
                tu = (tag[0], pos)
                l.append(tu)
        return l

def getVocabulary(matrix):
        s = set()
        for i in matrix:
                for j in i:
                        s.add(j)
        vocabulary = sorted(s)
        return vocabulary

###############################################
#                     LOGISTIC REGRESSION
###############################################
eps = 1e-9;
def le(a, b):
        return b-a > eps

# Distance between 2 vectors
# Return an scalar
def distance(A, B):
        aux = (A - B) * (A - B)
        sumV = np.sum(aux)
        return math.sqrt(sumV)

# Choose best Uk for vector Xi
# Return index of best centroide
def clusterCentroid(Xi, Uk):
```

```
        c = 0
        minDis = 1000000
        for i in range(0, len(Uk)):
                dis = distance(Xi, Uk[i]) # Escalar
                # print(dis)
                dis = (dis * dis)
                if le(dis, minDis):
                        minDis = dis
                        # print("Me quedo con: ", i)
                        c = i
        return c

# Get average of elements in the cluster
# Return a vector with the average of the elements in the cluster
def average(matrix, C, index):
        matrixAux = list()
        aux = np.zeros(shape = (len(matrix[0]))) # matrix[0] x 1
        matrixAux.append(aux);
        cont = 0
        for i in range(0, len(C)):
                if int(C[i]) == index:
                        cont = cont + 1
                        matrixAux.append(matrix[i])
        resultant = np.array(matrixAux)

        # Matrix that have all results for this index
        result = np.array(resultant)
        sumVector = result.sum(axis = 0)

        sumVector = sumVector / len(sumVector)
        return sumVector

# Cost Function
# Return an escalar that represent the cost function
def distortion(X, Uk, C):
        suma = 0
        for i in range(0, len(C)):
                v = distance(X[i], Uk[int(C[i])])
                v = v * v
                suma = suma + v
        suma = suma / len(C)
        return suma

def printResult(C, Y):
```

```
        #HAM = 1
        #SPAM = 0
        Uk1_HAM = 0
        Uk1_SPAM = 0
        Uk2_HAM = 0
        Uk2_SPAM = 0

        for i in range(len(C)):
                if int(C[i]) == 0:
                        if Y[i] == 1:
                                Uk1_HAM = Uk1_HAM + 1
                        else:
                                Uk1_SPAM = Uk1_SPAM + 1
                if int(C[i]) == 1:
                        if Y[i] == 1:
                                Uk2_HAM = Uk2_HAM + 1
                        else:
                                Uk2_SPAM = Uk2_SPAM + 1

        # if (Uk1_HAM > Uk2_HAM and Uk2_SPAM > Uk1_SPAM) or (Uk2_HAM >
        ↪  Uk1_HAM and Uk1_SPAM > Uk2_SPAM):
        print("Cluster 1: HAM:", Uk1_HAM, " SPAM:", Uk1_SPAM)
        print("Cluster 2: HAM:", Uk2_HAM, " SPAM:", Uk2_SPAM)

##############################################
##############################################
##############################################

# Get Tokens by Generate.txt to create dictionary of lemmas
#Read file spam/ham
fpathCorpus =
↪  '/Users/abiga/Desktop/AbiiSnn/GitHub/Natural-Language-Processing/Practice/23/corpus
code = 'ISO-8859-1'
corpus = getText(fpathCorpus, code)
tokens = getTokens(corpus)

#Matrix and Y
matrix = list()
auxY = list()
xi = list()
for token in tokens:
        if token != 'spam' and token != 'ham':
                xi.append(token)
        else:
```

```
                typeMessage = 0
                if token == 'ham':
                        typeMessage = 1
                auxY.append(typeMessage) #Create Y
                matrix.append(xi) #Create X
                xi = list()
Y = np.array(auxY)

#Tagging
matrixTag = list()
for i in range(0, len(matrix)):
        auxTag = tag(matrix[i])
        matrixTag.append(auxTag)

#Lemmatize
matrixLem = list()
wnl = WordNetLemmatizer()
for i in range(0, len(matrixTag)):
        l = list()
        for j in range(0, len(matrixTag[i])):
                lemma = wnl.lemmatize(matrixTag[i][j][0])
                t = (lemma, matrixTag[i][j][1])
                l.append(t)
        matrixLem.append(l)

vocabulary = getVocabulary(matrixLem)

# Sacar frecuencia y obtener matrix
vectors = getVectors(vocabulary, matrixLem)
frecuency = getFrecuency(vectors)

################################
#                K-Means Algorithm
################################
k = 2
n = len(frecuency)
m = len(frecuency[0])
Uk = np.zeros(shape = (k, m)) # k x m
C = np.zeros(shape = (len(frecuency))) # frecuencyTraining x 1

minCos = 1000000
iterations = 50
for iteration in range(0, iterations):
        # Initialize K cluster centroids
```

```
for i in range(0, k):
        r = random.randint(1, len(frecuency)-1)
        Uk[i] = frecuency[r] #Assign random Xi

for ite in range(0, 5):
        # Run K means algorithm:
        for i in range(0, len(frecuency)):
                C[i] = clusterCentroid(frecuency[i], Uk) # Return an
                 ↪  index between 0 and k-1

        for i in range(0, k):
                Uk[i] = average(frecuency, C, i) # Return vector

        print("-----------------------------------")
        printResult(C, Y)
        # Get distortion
        c = distortion(frecuency, Uk, C)
        print("Distortion:", c)
        if(le(c, minCos)): # Save best answer
                minCos = c
                Uk = auxUk
```