INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA

ESCUELA DE (INGENIERÍA Y TECNOLOGÍA – INGENIERÍA Y GESTIÓN - POSTGRADO)

¿De qué se habla cuando se habla de cerveza?

AUTOR/ES: Zenone Pedro Octavio. Legajo: 104250

DOCENTE/S TITULAR/ES O TUTOR/ES: Gustavo Arjones

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE ESPECIALISTA EN CIENCIA DE DATOS

BUENOS AIRES

PRIMER CUATRIMESTRE, 2019

Introducción 2

Estado de la cuestión 3

LDA 3

NMF 3

SNA 4

Planteo del problema 4

Solución 5

Extracción 5

Preprocesamiento 7

Topic modeling 7

SNA 11

Insights 11

Conclusiones y futuros trabajos 13

Bibliografía 13

Anexo: Código fuente 14

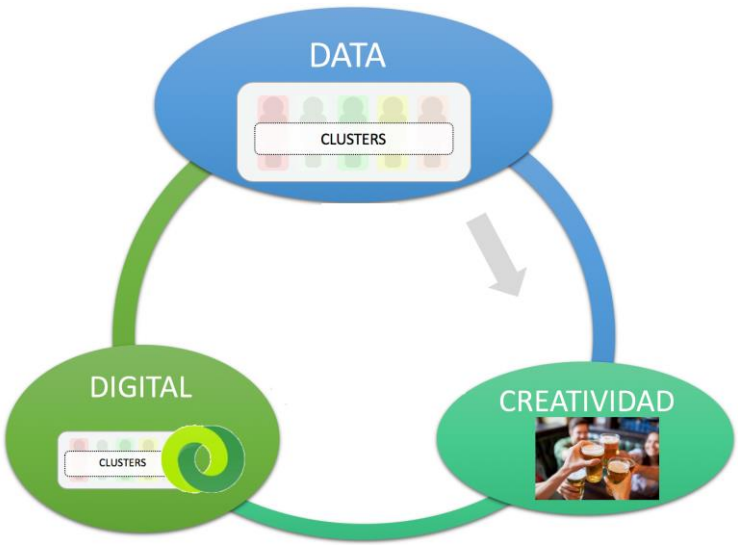
Introducción

La necesidad de generación e impacto de contenido personalizado es un requerimiento en auge que las marcas comienzan a exigir a las agencias de marketing. Esto quiere decir que cada persona contactable será impactada con un mensaje personalizado con el cual se sienta identificada. Existen diversas formas de generar audiencias que permitan segmentar a los usuarios en función de sus gustos o hábitos, pero estas dependen de la concepción de los datos: clicks en un sitio, items comprados o comentarios en redes sociales.

Para este caso de estudio se analizarán los comentarios de las redes sociales con palabras vinculadas directamente a la cerveza, ya que el cliente en cuestión es una cervecera. Con el fin de entender cuáles son los tópicos más relevantes se realizará una escucha en Twitter e Instagram, obteniendo comentarios, posteos y retweets relativos a la temática. Debido al volumen generado resulta inviable la lectura del contenido por personas, por eso aplicaremos técnicas de clustering que entiendan el lenguaje natural (NLU), agrupando de forma automática las palabras en conceptos.

Una vez obtenidos los clusters, se realizará un análisis de redes sociales (Social Network Analysis, SNA) graficando un mapa de relaciones (grafo) que permita entender cuál es la relación entre e intra tópico, permitiendo encontrar subcomunidades dentro de los tópicos.

Una vez realizado este entendimiento se concluirá con la generación de personas tipo que representen estos clusters (insumo para que el área creativa pueda generar sus piezas correspondientes a las temáticas detectadas), así como la asignación de cada persona (ID de usuario en red social) a un cierto cluster. De esta forma se logra cerrar el círculo Data science – Creatividad – Digital (look-alike + Impacto) [1][2]

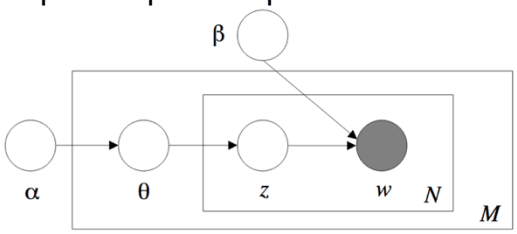


Estado de la cuestión

LDA

Latent Dirichlet Allocation es una técnica de tipo no supervisada que a partir del entendimiento del lenguaje natural permite asignar tópicos a cada documento. Dichos tópicos estarán conformados por un conjunto de palabras con una cierta probabilidad de pertenencia a dicho tópico.

LDA es un modelo probabilístico generativo de una colección de documentos hechos de palabras, también LSI (tf-idf + SVD) bayesiano. El proceso de entrenamiento consiste en asignar a cada palabra una probabilidad de pertenencia a un tópico (N palabras w al tópico z) según la distribución de Dirichlet, parametrizada por alpha y beta según corresponda. A su vez se asigna de forma aleatoria una probabilidad de partencia de documento a tópico (M documentos, con distribución theta). De esta forma se generan dos probabilidades condicionales P(Topic | Word) y P(Document | Topic) que se resumen en la multiplicación de ambas para explicar la probabilidad de dicha palabra al tópico.



El proceso de cálculo de las probabilidades P(Topic | Word) y P(Document | Topic) es algo iterativo ya que el mismo se va actualizando (filosofía bayesiana).

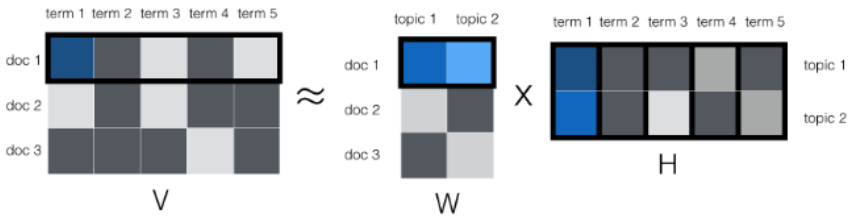
Los supuestos fuertes que este modelo toma son:

- K tópicos como premisa, es decir que dada una cierta cantidad de tópicos el modelo generará la segmentación
- Cantidad de palabras a tomar.

Los otros dos parámetros tuneables son alpha, altos valores hacen que el documento este compuesto por muchos tópicos, y beta, altos valores hacen que los tópicos estén conformados por varias palabras.

NMF

Non matrix factorization es una técnica de modelado del algebra lineal donde mediante una factorización de la matriz inicial (documento vs palabras con la frecuencia como contenido, V) en dos matrices W y H, de forma tal que V = W.H, ambas con coeficientes positivos. En esta técnica de descomposición se reduce la dimensionalidad de forma tal que el rango de la matriz W.H sea de k (número de tópicos), optimizando la norma: $\|V - WH\|^2$



Como puede verse H representa la importancia de cada palabra en dicho tópico y W representa la importancia del documento por tópico.

SNA

El social network analysis es una técnica que permite explicar las relaciones entre personas a través de un grafo. Este se construye generando uniones entre nodos (personas) a través de aristas (mensajes). Sobre este gráfico se pueden realizar diversas métricas que permiten entender cuál es el poder de dichas personas en la red estudiada y que influencia poseen en la misma. No obstante, para este caso de estudio se realizará únicamente una inspección gráfica a modo de entendimiento de relación entre comunidades.

La disposición de los nodos se genera a partir de la técnica de Force Atlas2, esta consiste en generar una ecuación de armonía símil a la de un sistema físico donde tenemos fuerzas de atracción F = -kd (resorte) que se traduce a F(n1,n2) = d(n1,n2), repulsión $F = -\frac{k^2}{d}$ (repulsión de cargas eléctricas iguales) que se implementa en redes sociales como $F(n1, n2) = \frac{(n1)+1)(n2)+1)}{d(n1,n2)}$ y una constante (gravedad) F(n) = k (deg(n) + 1). Dichas fuerzas están parametrizadas por una constante la cual puede ser tuneada. La convergencia de la suma de todas las fuerzas igualadas a cero representa el equilibrio de la ecuación.

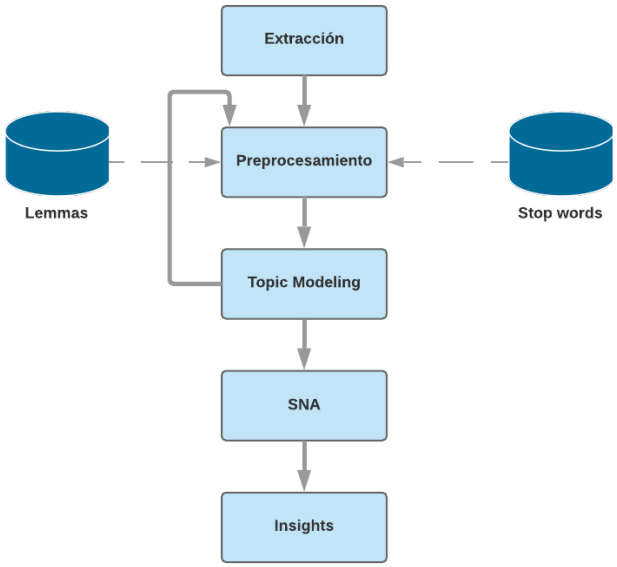
Nodos cercanos entre sí serán aquellos similares los cuales se agruparán en clusters interpretables como comunidades (personas afines, hemofilia).

Planteo del problema

Todo proyecto surge como solución a una problemática determinada, en este caso el área creativa requería entender de forma cualitativa cuáles eran los temas latentes para la temática cerveza. Con estos insights ellos desarrollaron piezas visuales específicas con las cuales las personas a ser impactadas se sienten más identificadas con la marca, de esta forma se logra aumentar la consideración hacia la marca (por más que no conozca la marca la considerará por haber visto publicidad con la que se sintió identificado).

Solución

A continuación, se muestra el flujo de trabajo adoptado para este proyecto, típico en proyectos de data science donde contamos con una etapa de extracción y almacenamiento de algunas fuentes de información, luego a estos datos extraídos se les aplica transformaciones necesarias para pasar a la etapa de modelado. En los proyectos de tipo descriptivos, la etapa final consiste en la interpretación de los datos para dar recomendaciones o entendimientos descubiertos que generen valor al negocio.



Extracción

En esta etapa se usa una herramienta de extracción de tweets y mensajes de Instagram (Brandwatch) la cual permite a partir de una cierta búsqueda obtener tweets, retweets, comentarios y hashtags de Instagram. Los datos que esta plataforma entrega son:

- Fecha del mensaje
- Author (Id y screen_name)
- Texto
- Url del mensaje
- Fuente de donde proviene (Twitter o Instagram)
- Menciones que se realizan en el mensaje
- Geolocalización
- Características del usuario (followers, Friends, comentarios, etc)

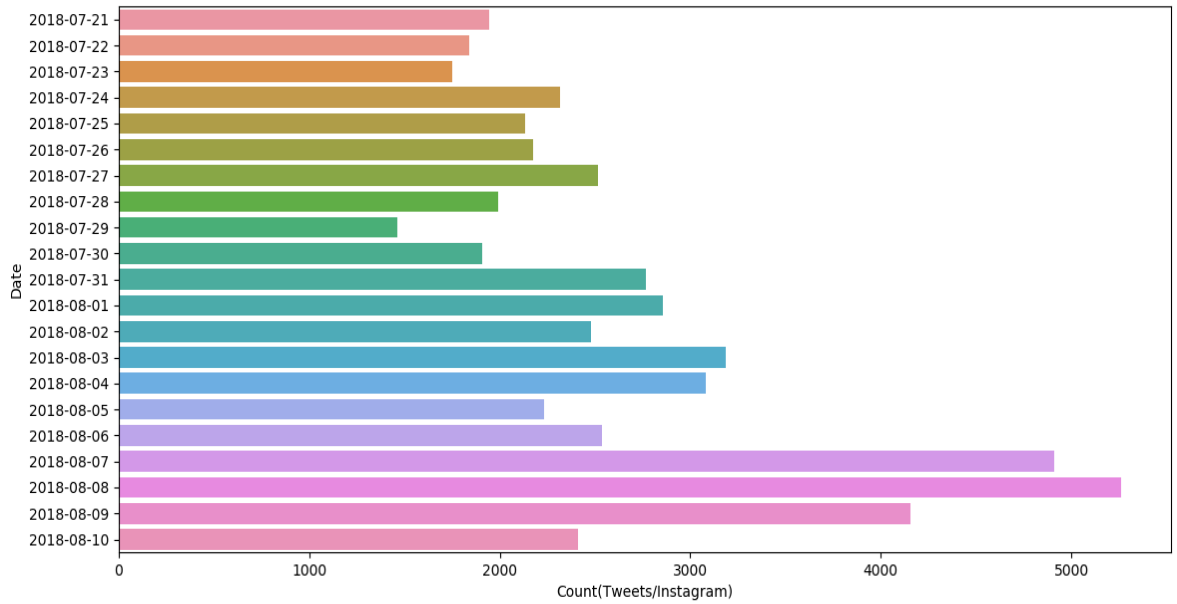
La búsqueda realizada fue:

country:ar
AND
*((at_mentions:schneiderarg OR (schneider OR s*neider OR es*neider) OR (cerveza OR birra) OR (amig* OR amistad) OR (musica OR music OR rock) OR (morfi OR comida OR sali* OR gastronomia) OR (futbol OR fubol OR football) OR ((cerveza OR birra)NEAR/8(amig* OR amistad OR ameo* OR amea* OR "dia del amigo" OR pibes)) OR hashtags:amigos OR hashtags: amigas) OR (((cerveza OR birra)NEAR/8(comida OR gastronomia OR morfi OR comer OR brunch OR hamburguesa OR burger OR pizza)) OR hashtags:foodporn) OR ((((cerveza OR birra)NEAR/8(musica OR rock OR dj OR vivo OR music*)) OR hashtags:music) OR hashtags:oktoberfest OR hashtags:beer OR*

hashtags:diainternacionaldelacerveza) OR (((cerveza OR birra)NEAR/8(futbol OR football OR futball OR fubol OR canchita OR juego OR pelota OR "papifutbol" OR "papi futbol" OR "papi fubol" OR "papi" OR "papi fútbol")) OR hashtags:futbol)OR (((cerveza OR birra)NEAR/8("futbol femenino" OR "female football" OR "futbal femenino" OR "fubol femenino" OR "mamifutbol" OR "mami futbol" OR "mami fubol" OR "mami fútbol" OR "torneo" OR "torneito")) OR hashtags:futbolfemenino))
site: (twitter OR Instagram OR facebook)
NOT
(tenemos OR nuestras OR raw:hs OR presentamos OR invitamos OR "para todos" OR veni OR vengan OR promo OR probarla OR festejamos OR queres OR esperamos OR $ OR "2x1" OR raw:#follow4follow OR raw:#likeforlike OR "viv? la experiencia")*

La escucha se realizó para 1 semana, obteniéndose un volumen de 55 mil tweets y retweets.

De forma exploratoria se analiza la cantidad de comentarios por día detectando y eliminando los retweets espurios que aparecieron. Finalmente, la serie quedó como se muestra a continuación.



Esta misma búsqueda podría ser simulada desde la API de twitter, pero esta plataforma me permite realizar esta tarea de forma mucho más sencilla, sin tener que preocuparse por el desarrollo de la conexión o almacenamiento de los datos obtenidos.

Desde luego que se tenía conocimiento previo del mercado, es por ello que se agregaron palabras claves para enriquecer la búsqueda. La idea de la técnica aplicada es el de poder encontrar nichos desconocidos para el área de investigación de mercado. No obstante, al cargar con palabras claves que aumenten el volumen de datos puede que traigan con sigo sesgos asociados.

Preprocesamiento

Como etapa previa al modelado, el preprocesamiento es una etapa vital para poder generar el agrupamiento de palabras. Este consiste en llevar todas las palabras a un mismo formato (transformar todas las palabras a minúscula y quitarles los acentos ya que en las redes sociales se cometen muchas faltas de ortografía), quitar los símbolos que para este caso no representan nada (los hashtags tienen un gran significado por lo que se quita el # y luego se buscan palabras contenidas en ese hashtag, ejemplo futbolamigos ->), las menciones a otros usuarios, aquellas palabras de alta frecuencia que no explican nada (si, el, yo, también, etc también conocidas como stop-words) y lo más importante llevar todas las palabras a una misma raíz (stemming).

Si bien stemming consiste en truncar la palabra (ejemplo: comer -> com), la lematización lleva efectivamente las palabras a una misma raíz pero realizando un entendimiento de la palabra en su contexto, por ejemplo: “esto como que no funciona” o “como todo el día”. En un caso “como“ se refiere al verbo comer, mientras que en el primer caso no es un verbo. Este proceso de entendimiento no fue realizado, sino que se usó un diccionario de lemas español, el cual se fue enriqueciendo también donde cada palabra tiene mapeado su lema.

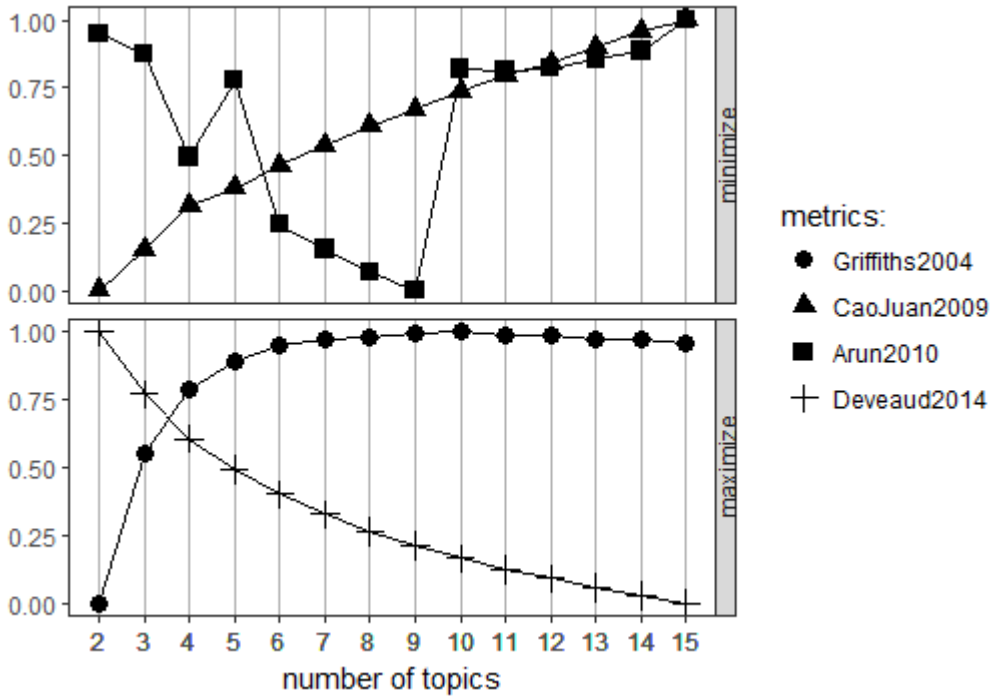
De esta forma los pasos involucrados quedan definido de la siguiente forma:

Con el fin de encapsular esta etapa en un objeto, se crea la clase LDA_preproc la cual contiene los métodos necesarios para transformar y explorar el texto input dejándolo listo para la etapa de modelado.

Topic Modelling

Una vez realizada la limpieza y transformación de los datos se realiza la tokenización de las frases, esto consiste en tomar las frases como si fuesen una bolsa de palabras, con una frecuencia de aparición asociada, llamada corpus el cual hará referencia al diccionario de palabras generado.

Luego se procede a calcular la cantidad de tópicos óptima. Para ello se calculan diversas métricas (Griffiths2004, CaeoJuan2009, Arun2010, Deveaud2014) que explican la coherencia de los tópicos generados reiteradas veces. La cantidad óptima, de forma aproximada ya que es uno el que debe interpretarlo, será aquella en donde se genere algún cambio en la curvatura de la gráfica.



Según el análisis realizado, la cantidad de tópicos interpretable resultó de 4-6. Con esto ya se puede comenzar con la etapa de modelado (LDA), siendo el único hiperparámetro la cantidad de nodos. Esta implementación puede encontrarse en lda_topicOpt.R.

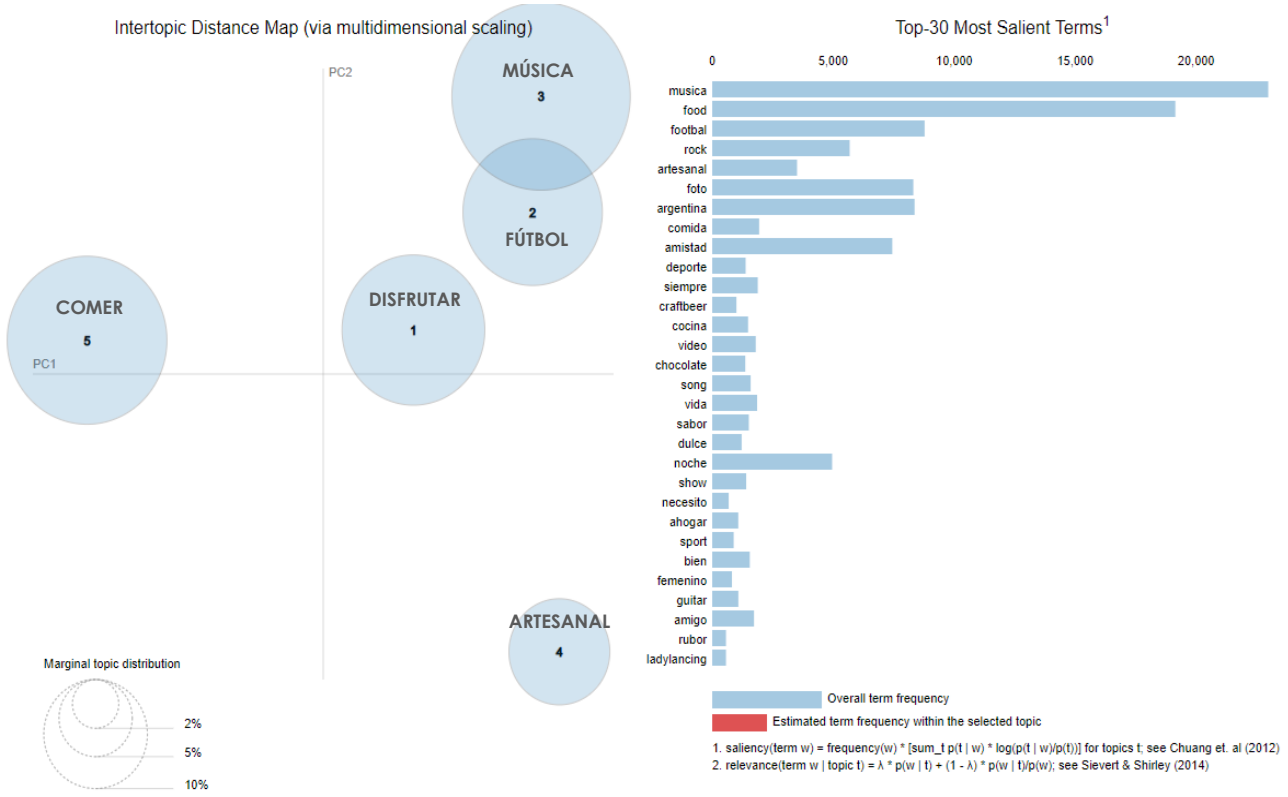
Con el fin de realizar un entendimiento de los clusters encontrados se generó una visualización que proporciona el paquete pyLDAvis, el cual realiza una reducción de dimensionalidad a 2, manteniendo la distancia entre tópicos.

Las características a tener en cuenta son:

- El diámetro de los círculos representa el volumen de documentos que contiene el cluster.

- Las 30 palabras más relevantes del tópico, donde la barra en azul representa la cantidad de veces que aparece la palabra en el universo, mientras que las rojas muestran la cantidad de veces que aparecieron en el tópico.

- El espacio entre tópicos permite entender la relación de las palabras entre ellos. Tópicos cercanos significa que se usan palabras similares en ambos, es decir que tienen una relación como se puede observar entre música y fútbol.



La caracterización de cada cluster se realiza interpretando palabras y poniéndolas en contexto realizando un muestreo de las frases pertenecientes a dicho tópico que contiene esa palabra.

DISFRUTAR (0, '0.022*"amistad" + 0.009*"siempre" + 0.008*"noche" + 0.006*"bien" + 0.004*"casa" + 0.004*"artesanal" + 0.004*"tomo" + 0.004*"vino"')

FÚTBOL (1, '0.053*"footbal" + 0.019*"argentina" + 0.008*"deporte" + 0.005*"sport" + 0.005*"femenino" + 0.005*"corazon" + 0.004*"copa" + 0.004*"amistad"')

MÚSICA (2, '0.085*"musica" + 0.026*"foto" + 0.021*"rock" + 0.015*"argentina" + 0.011*"noche" + 0.007*"buenosaires" + 0.007*"video" + 0.006*"song"')

LIFESTYLE (3, '0.026*"artesanal" + 0.010*"craftbeer" + 0.009*"vida" + 0.008*"necesito" + 0.006*"rubor" + 0.006*"gratiiii" + 0.006*"maquillado" + 0.006*"ladylancing"')

COMER (4, '0.088*"food" + 0.009*"comida" + 0.007*"sabor" + 0.007*"cocina" + 0.006*"chocolate" + 0.006*"piza" + 0.006*"amistad" + 0.006*"dulce"')

A partir de las probabilidades de cada palabra por tópico, se entiende que:

Cluster 1. Las palabras amistad, noche, casa hacen referencia al disfrute y las salidas de amigos

Cluster 2. Las palabras footbal, deporte, copa hacen referencia la Futbol.

Cluster 3. Las palabras como músca, rock, song hacen referencia a la música.

Cluster 4. Las palabras Artesanal, Craftbeer hacen referencia a la cerveza artesanal.

Cluster 5. Las palabras food, comida, cocinar hacen referencia a la comida.

ARTESANAL

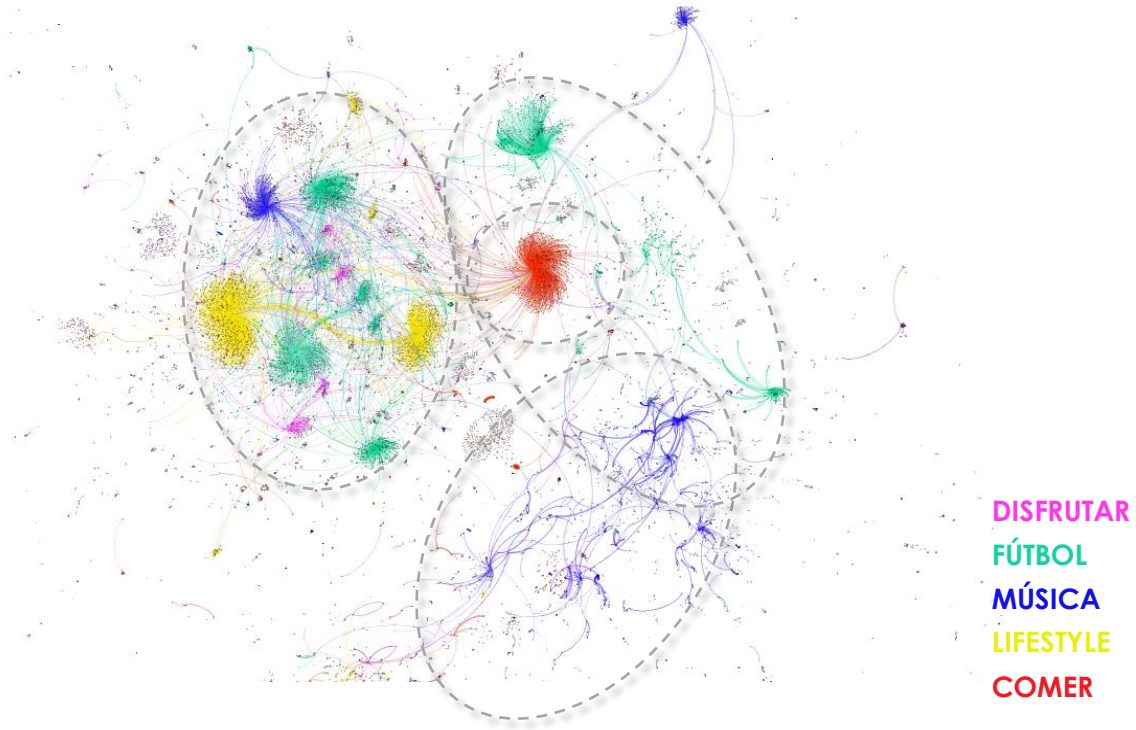
AMISTAD

Las nubes fueron generadas con el archivo wordCloudGenerator.R

SNA

Tomando los comentarios, retweets y menciones se generó un grafo de relaciones entre personas, donde se le asignó a cada persona un tópico en particular (a cada mensaje se le asignó un tópico y luego se tomó la moda agrupada por usuario).

Para poder estudiar de forma gráfica las conexiones se realizó una visualización usando Gephi donde cada nodo representa una persona y el color del mismo el tópico de pertenencia. La distancia entre nodos esta dada por la ecuación de equilibrio de Force Atlas 2 donde aquellos que estén muy separados significa que no están relacionados y en caso que estén juntos es debido a que son similares.



Mediante este gráfico se puede hacer un entendimiento de las comunidades que se generan. Por un lado puede verse una comunidad aislada de música representada por bandas de rock o alternativas que hablan entre sí, junto con seguidores que comentan y siguen a las mismas.
La comida resulta un nexo entre fútbol y el cluster artesanal, mientras que el cluster de disfrute se encuentra más vinculado con el fútbol.

El código para generar las tablas de relaciones se encuentra en NodeAristas.py

Insights

Mediante las técnicas anteriores aplicadas se realizó un entendimiento del universo Cerveza, generando perfiles tipos tipos de personas.

Los usuarios identificados en el cluster *Disfrutar* conversan de la temática sin vincularse con otros tópicos. Sin embargo, aquellas personas dentro de los clusters *Artesanal*, *Comer* y *Fútbol* hacen referencia a algunos temas que pertenecen al grupo *Disfrutar*. Es decir, aquellos que hablan de la diversión sólo hablan de la temática, en cambio aquellos que tocan temas como cerveza artesanal, comer o fútbol pueden involucrar a la diversión.

La cercanía entre los clusters *Fútbol* y *Música* da cuenta de las similitudes en las formas de expresarse de ambos grupos. Puede estar dado por similitudes la pasión que generan ambos géneros.
El tópico *Artesanal* representa un nicho pequeño dentro del rubro cerveza. Está compuesto por un grupo social con intereses premium, ej: cerveza Corona, Netflix.

Música es un tópico con mucha dispersión. Está compuesto por una gran cantidad de nodos primarios con conversaciones locales (shows de bandas under, fiestas). Estas conversaciones tienen pocos vínculos con otros tópicos, se conecta levemente con *Disfrutar*, *Comer* y *Fútbol*.

El tópico *Fútbol* invita a otros tópicos a formar parte de su conversación. En su radio interceden *Disfrutar*, *Música* y *Comer*.

Conclusiones y futuros trabajos

Se ha desarrollado un pipeline de trabajo capaz de entender el lenguaje natural segmentando grandes volúmenes de conversaciones provenientes twitter e instagram en tópicos que permiten comprender ¿de qué se habla cuando se habla de un producto?.

Durante este proceso se han dejado los retweets debido a que esta es una unidad de expresión de pensamiento (la persona leyó el tweet de otro, lo procesó, lo entendió y lo compartió), generando así un volumen diario elevado.

Los tópicos encontrados tienen un correlato directo hacia la marca, permitiéndole entender hacia que nichos apuntar, cuáles deben ser los mensajes a enviar y quiénes son esas personas afines a los mensajes.

Las mejoras a tener en cuenta a futuro son:

- Replicar el mismo flujo de trabajo usando técnicas de NMF (non-negative matrix factorization), la cual es ampliamente usada junto con LDA, con el fin de ver cual entrega resultados más coherentes.

- Sacar las palabras que tengan frecuencia menor a 2 (palabras sin relevancia que hacen que el modelo se demore más en finalizar y puede llevar a que los clusters resultantes difieran para semillas distintas, esto no se hizo ya que se debía agregar código paralelizable insumiendo tiempo de desarrollo).

- Correr LDA varias veces con distintas semillas identificando un cluster promedio mediante comparación vectorial y la importancia promedio junto con su desvío de cada palabra contenida en el cluster promedio. Esta última opción requeriría desarrollar una visualización adhoc que tome como input los resultados calculados.

- Remover algunas stopwords que quedaron pendientes.

- Agregar datos demográficos de los usuarios, con el fin de ver si el sexo tiene relevancia en los clusters o si realizando este proceso por sexo dan clusters distintos.

Bibliografía

[1] https://adexchanger.com/social-media/twitter-rolls-out-lookalike-audiences-new-mobile-ad-ids-targeting-by-phone-number/

[2] https://www.facebook.com/business/a/lookalike-audiences

[3] Latent Dierichlet Allocation. Andrew Ng, David Blei. Journal of Machine Learning Research 3 (2003) 993-1022

[4] https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/

[5] https://medium.com/@lettier/how-does-lda-work-ill-explain-using-emoji-108abf40fa7d

[6] https://en.wikipedia.org/wiki/Non-negative_matrix_factorization

[7] Maksim Tsvetovat. Social Network Analysis for Startups O'Reilly 2011

[8] http://derekgreene.com/slides/derekgreene_gephi_slides.pdf

Anexo: Código fuente

- *El código puede encontrarse en mi github:*
  - https://github.com/pedroZenone/TopicModelling

- *Las visualizaciones del resultado:*
  - https://pedrozenone.github.io/TopicModeling/
  - https://pedrozenone.github.io/SNA/

```
import nltk
from nltk import word_tokenize
from nltk.stem import SnowballStemmer
from nltk.corpus import stopwords
from string import punctuation
import pandas as pd
from unicodedata import normalize
import re
#import spacy
from gensim import corpora, models, similarities
import gensim
import os
import pyLDAvis.gensim
import seaborn as sns
from sklearn.feature_extraction.text import CountVectorizer
import itertools
import matplotlib.pyplot as plt
import numpy as np
# para distribuido: pero no funciona, no hace falta instalar!
#conda install pytorch -c pytorch
#pip3 install torchvision
# pip install ipykernel
# pip3 install pyro-ppl

# data format: Must be a dataframe with columns ["Full Text","Mentioned Authors","Date"]
class LDA_preproc:

    def __init__(self,dataSource,lstopWords = [],verbose = 0):

        self.data = dataSource
        self.verbose = verbose

        # levanto las stop words
```

```
        self.my_stopwords = stopwords.words('spanish') + stopwords.words('english') +
lstopWords + ['RT', 'rt']

        # Creo el diccionario del lematizer!
        self.lemmaDict = {}
        with open('lemmatization-es_v2.txt', 'rb') as f:
          data = f.read().decode('utf8').replace(u'\r', u'').split(u'\n')
          data = [a.split(u'\t') for a in data]

        with open('lemmatization-es_add_v2 .txt', 'rb') as f:
          data_2 = f.read().decode('utf8').replace(u'\r', u'').split(u'\n')
          data_2 = [a.split(u'\t') for a in data_2]

        data = data+data_2  # uno los dos diccionarios y cargo las keys con valor

        for a in data:
          if len(a) >1:
            self.lemmaDict[a[1]] = a[0]

        if(verbose > 0):
              print("Lemma dict Uploaded")

    def my_lemmatizer(self,word):
       return self.lemmaDict.get(word, word)

    # Las letras que estan repetidas de forma consecutiva las pasa a una sola, salvo las
que en ingles tiene sentido que esten dobles
    def repetidos(self,x):
        y = x
        abc = [x for x in 'abcdfghjkpqvwxyzui']   # saco la ll y rr
        for letra in abc:
            y = re.sub(r''+letra+'{2,}',letra,y)
        y = re.sub(r'l{3,}','ll',y)
        y = re.sub(r'r{3,}','rr',y)
        y = re.sub(r'e{3,}','ee',y)
        y = re.sub(r'o{3,}','oo',y)
        y = re.sub(r's{3,}','ss',y)
        y = re.sub(r'n{3,}','nn',y)
        y = re.sub(r't{3,}','tt',y)
        y = re.sub(r'm{3,}','mm',y)
        return y

    # borra la linea donde este una palabra en especifico
    def delete_containedWord(self,y):
        indexes = [i for i,x in enumerate(self.texts) if(not re.search(r'\b'+y+r'\b',x))]
        self.texts = [self.texts[x] for x in indexes]
        self.token = [word_tokenize(x) for x in self.texts]
        self.tweets = [self.tweets[x] for x in indexes]
        self.data = self.data.iloc[indexes,:]

    # borra los tweets donde esta mencionado un cierto autor
    def delete_MentionedAuthor(self,x):
```

```
        self.data = self.data.reset_index(drop = True)
        self.data = self.data.loc[self.data["Mentioned Authors"] != x]
        indexes = self.data.index.tolist()
        self.texts = [self.texts[x] for x in indexes]
        self.token = [self.token[x] for x in indexes]
        self.tweets = [self.tweets[x] for x in indexes]

    def acentos_out(self,s):
        x = re.sub(
            r"([^n\u0300-\u036f]|n(?!\u0303(?![\u0300-\u036f])))[\u0300-\u036f]+", r"\1",
            normalize( "NFD", s), 0, re.I )
        return x

    # lower -> delete @user -> delete ascentos -> remove non characters -> standarized
repeated -> delete social links -> lemmatize
    def tokenize(self,text):
        text=text.lower()
        text = re.sub(r'\B@\S*\s?','',text)  # le saco el @algo
        text = self.acentos_out(text)
        text = ''.join(re.findall(r'[a-z\s]',text)) # le saco los caracteres que no sean words ni
numeros
        text = self.repetidos(text)
        text = re.sub(r'\w*(twiter|youtube|facebook|instagram|bitly)\w*','',text)  # le saco los
que son propagandas de twitter, facebook o youtube
        tokens =  word_tokenize(text)
        tokens = [x for x in tokens if(x not in self.my_stopwords)]
        tokens = [self.my_lemmatizer(x) for x in tokens]
        return tokens

    # Main function to start preprocessing the data. This method generates 3 ouputs: the
raw tweets,
    # tokenized preproc tweets and preproc tweets
    def preprocessing(self):
        self.tweets = self.data["Full Text"].as_matrix().tolist()
        self.token = [self.tokenize(x) for x in self.tweets]
        self.texts = [' '.join(x) for x in self.token]

        if(self.verbose > 0):
            print("Data Preprocesada. Para obtener los tweets crudos: get_rawTweets()")
            print("Para obtener los tweets procesados en Tokens: get_procTokenTweets()")
            print("Para obtener los tweets procesados en text : get_procTextTweets()")

    def get_rawTweets(self):
        return self.tweets

    def get_procTokenTweets(self):
        return self.token

    def get_procTextTweets(self):
        return self.texts

    def get_Data(self):
```

```
        return self.data

    # Te grafica la cantidad de tweets en el tiempo
    def exploratoryPlot(self):
        analysis = pd.to_datetime(self.data["Date"], format='%Y-%m-%d', errors='coerce')
        analysis = analysis.apply(lambda x: str(x.year) + '-' + str(x.month).zfill(2) + '-' +
str(x.day).zfill(2))
        GB=pd.DataFrame(analysis.groupby(analysis).count())
        GB.columns = ["Count"]
        GB = GB.reset_index()
        ax = sns.barplot(x = "Count",y = "Date",data = GB)
        ax.set( ylabel = "Date",xlabel="Count(Tweets/Instagram)")
        plt.show()

    # En caso de querer recargar las stopwords y no tner que reprocesar todo devuelta!
    def update_StopWords(self,lStopWords):

        self.token = [[x for x in y if(x not in lStopWords)] for y in self.token]
        self.texts = [' '.join(x) for x in self.token]

#
==========================================================================
=========
# @brief: Para entender cual es el largo necesario donde cortar y decir que una palabra
# es larga grafico un histograma de longitudes, de ahi se puede calcular el treshold
#
# @ param: thresh. Si la palabra tiene un largo superior a tresh entrega la palabra
# @param: hist. Si vale 0 no muestra la grafica y entrega todas las palabras que superan
thresh
#             Si vale 1 se muestra el histograma. De este grafico se saca por inspecion el
tresh
#
# @out: palabras que tienen un largo superior a tresh
#
==========================================================================
=========

    def get_potencialLotWords(self,hist=0,tresh = 15):

        self.tresh = tresh

        def do_nothing(tokens):
            return tokens

        vectorizer = CountVectorizer(tokenizer=do_nothing,
                                     preprocessor=None,
                                     lowercase=False)

        vectorizer.fit_transform(self.token)  # a sparse matrix
        vocab = vectorizer.get_feature_names()  # a list

        if(hist == 1):
```

```
        lplt = [len(x) for x in vocab]  # me armo una lista para graficar la distribucion de
largos
        plt.hist(lplt, bins = np.arange(min(lplt),max(lplt),1))

    return [x for x in vocab if(len(x) >= tresh)]


  def truncator(self,x,pattern,lPattern):

    if(pattern.search(x)):
      token = word_tokenize(x)

      aux = [[ k  for k in lPattern if((k in y) & (len(y) >= self.tresh))] for y in token]   # las
palabras long las va a poner con sus posibles combinaciones
      auxx= [x if(len(x) > 0) else [token[i]] for i,x in enumerate(aux)] # las que no son
plabras long quedaban vacias, entonces las relleno con esta sentencia
      flatten = list(itertools.chain.from_iterable(auxx)) # hago flat la lista

      return ' '.join([self.my_lemmatizer(x) for x in flatten] ) # la vuelvo a pasar por el
lemmatizer y la transformo en texto

    return x

#
=========================================================================
=========
# @brief: este metodo busca si la palabra esta contenida dentro dentro de los tweets
# ejemplo futbol -> futbolamigos True
#
# @param: lLong. Lista de palabras a ver si esta contenida.
#
=========================================================================
=========
  def truncateLongWords(self,lLong):

    pattern = re.compile(''.join(['|'+ y for y in lLong])[1:])
    self.texts = [self.truncator(x,pattern,lLong) for x in self.texts]
    self.token = [word_tokenize(x) for x in self.texts]

  # entrega la cantidad de veces que aparece cada palabra
  def countVectorizer(self):
    def do_nothing(tokens):
      return tokens

    vectorizer = CountVectorizer(tokenizer=do_nothing,
                    preprocessor=None,
                    lowercase=False)

    dtm = vectorizer.fit_transform(self.token)  # a sparse matrix
    vocab = vectorizer.get_feature_names()  # a list
    words_freq = np.asarray(dtm.sum(axis=0)).T
```

```
        DataFinal = pd.DataFrame([],columns = ["word","frequency"])
        DataFinal["word"] = vocab
        DataFinal["frequency"] = words_freq

        return DataFinal

    # hago una busqueda de las palabra find en todos los tweets. find es un regex patter a
buscar.
    def inspeccion(self,find):
        self.data = self.data.reset_index(drop= True)
        indexes = [i for i,x in enumerate(self.texts) if(len(re.findall(find,x)) > 0)]
        resu = pd.DataFrame([],columns = ['indiceRaw','mensage'])
        resu['mensage'] = [self.tweets[i] for i in indexes]
        resu['indiceRaw'] = indexes
        urls = self.data.Url.values.tolist()
        resu["url"] =  [urls[i] for i in indexes]
        return resu

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

######  1.Levanto data input y stop words (debe ser ua lista)

# Seteo directorio de trabajo
fileDir = os.path.dirname(os.path.realpath('__file__'))
fileIn= os.path.join(fileDir, 'input')

# Levanto la data y la limpio un poco
data = pd.read_excel(os.path.join(fileIn,"Cerveza.xlsx")
data = data[["Date","Url","Author","Page Type","Mentioned Authors","Full Text","Full
Name","Country Code","Resource Id"]]


######  2. Instancio la clase, inicializo
data_preproc = LDA_preproc(data,[])
data_preproc.preprocessing()  # arranco el preprocasamiento!

# me quedo con los tokens que encontre por el momento y la data cruda
token = data_preproc.get_procTokenTweets()
rawTexts = data_preproc.get_rawTweets()

######  3. Me fijo el historico de la data
#data_preproc.exploratoryPlot()

######  4. Me fijo como es la distribucion del largo de palabras y donde considerar un
palabra larga
palabrasLargas = data_preproc.get_potencialLotWords(hist = 0,tresh = 15)  # si hist = 0 no
muestra grafico

# cargo la lista de palabras que vi potables para truncar
lLargo = pd.read_excel("long.xlsx")['palabras'].tolist()  # me quedo con la raiz de las
palabras. Ej: amigosfutbol  -> amigos futbol
```

```
data_preproc.truncateLongWords(lLargo)

new_texts = data_preproc.get_procTextTweets()  # ahora voy a tener nuevos texts
token = data_preproc.get_procTokenTweets()

######  5. Analizo las frecuencias de las palabras. Aca puedo ver si tengo palabras que no estan en lematizer o si tengo stopwords a remover

Counter_df = data_preproc.countVectorizer()

######  6. Una vez identificadas las top words o palabras de frecuencia 1 que no aportan o de alta frecuencia que no digan nada:
new_stopwords = pd.read_excel("StopWords2.xlsx")['palabras'].tolist()
data_preproc.update_StopWords(new_stopwords )

token = data_preproc.get_procTokenTweets()

######  7. Inspecciono las palabras que me llaman la atencion. Puedo ver el mensaje donde aparecieron.

find = data_preproc.inspeccion(r'\bcomer\b')

# si queres hacer zoomin en una cierta palabra y hacer un topico sobre eso:
#data_1 = data_preproc.get_Data()
#data_1.loc[find.indiceRaw.values.tolist()].to_excel("Panal.xlsx")

#####  8.Si veo algo fuera d elugar lo puedo borrar:

data_preproc.delete_containedWord('guillermoprieto')
data_preproc.delete_containedWord('hamburgues')
data_preproc.delete_containedWord('sinaloa')
data_preproc.delete_containedWord('articulo')
data_preproc.delete_containedWord('gobernador')
data_preproc.delete_containedWord('gobierno')
data_preproc.delete_containedWord('gobernar')
data_preproc.delete_containedWord('edomex')
data_preproc.delete_containedWord('policia')
data_preproc.delete_containedWord('diadelamadre')
data_preproc.delete_containedWord('ley')
data_preproc.delete_containedWord('fiscal')
data_preproc.delete_containedWord('foodporn')
data_preproc.delete_containedWord('radiar')
data_preproc.delete_containedWord('taxista')
data_preproc.delete_containedWord('adn')
data_preproc.delete_containedWord('infomigra')
data_preproc.delete_containedWord('escrache')
data_preproc.delete_containedWord('ud')
data_preproc.delete_containedWord('adolescente')
data_preproc.delete_containedWord('laprida')

#####  9. Borro algun author que todos mencionan
data_preproc.delete_MentionedAuthor('@camiirodrigues_')
```

```
#####  10. Genero la base para R. Go to R:
texts = data_preproc.get_procTextTweets()
pd.DataFrame(texts,columns = ["data"]).to_csv("Go2R.csv",sep = ";",encoding = "UTF-8")

######  11. LDA. OJO, REFRESCAR token!!!:

token = data_preproc.get_procTokenTweets()  # siempre refresco la data antes de correr
todo el LDA!

# inic corpus 4 LDA
dictionary = corpora.Dictionary(token)
corpus = [dictionary.doc2bow(text) for text in token]
tfidf = models.TfidfModel(corpus) # step 1 -- initialize a model
corpus_tfidf = tfidf[corpus]  # step 2 -- use the model to transform vectors

for i in range(15):

    # en la gráfica de topicos, marca como el optimo 6 clusters
    lda = models.LdaModel(corpus, id2word=dictionary, num_topics=5,random_state  = 5*i)
    lda_display = pyLDAvis.gensim.prepare(lda, corpus, dictionary, sort_topics=False)
    pyLDAvis.save_html(lda_display, 'TopicModelling'+str(i)+'.html')
    print("Visualizacion ",str(i), " generada")
#pyLDAvis.show(lda_display)

# Si te gusta como queda el modelo, guardalo por las dudas ;)
fname = 'Topics'
lda.save(fileDir + '/modelos/'+fname)
lda = models.LdaModel.load(fileDir + '/modelos/'+fname)

##############################################
#num_topics = 6
#finaldf = pd.DataFrame([],columns = ["palabras","prob"])
#for num,l in enumerate(aux):
#
#    df_aux = pd.DataFrame([],columns = ["palabras","prob"])
#
#    for topicId in range(num_topics):
#        auxx =  pd.DataFrame(l.show_topic(topicId,topn=5),columns = ["palabras","prob"])
#        auxx["topic"] = topicId
#        df_aux = df_aux.append(auxx,ignore_index = True)
#
#
#    df_aux["it"] = num
#    finaldf = finaldf.append(df_aux,ignore_index = True)
#############################################################################

#####  12.  Le asigno un topico a cada tweet (debe tener una asociacion fuerte, sino le
pongo -1). Luego puedo hacer analisis subtopicos!

lda = models.LdaModel(corpus, id2word=dictionary, num_topics=5,random_state  = 5*14)
# una vez que elegi cual de los 8 clusters voy a querer usar me vuelvo a armar el modelo
```

```
def num_topic(x,tresh = 0.5):
    aux = lda.get_document_topics(x)
    rate = aux[np.argmax(([y[1] for y in aux ]))]

    umbral_score= tresh
    if(rate[1] > umbral_score):
        return rate[0]          # si el topico elegido es represetativo, devuelvo el valor
    else:
        return -1               # caso contrario deveulvo -1 para mostrar que no es relevante

# si queres ser poco restrictiva y agregar todas las palabras ->  tresh = 0
topic = [num_topic(x) for x in corpus]  # corpus esta definido arriba

reutilizar = data_preproc.get_Data()
reutilizar["Topic"] = topic
reutilizar["Lema Text"] = data_preproc.get_procTextTweets()
#retulizar = reutilizar.loc[reutilizar.Topic == 1]  # aca le pones el topico que te quieras traer

# Para guardalo:
writer = pd.ExcelWriter(r'DataWithTopic.xlsx', engine='xlsxwriter',options={'strings_to_urls':
False})
reutilizar.to_excel(writer)
writer.save()
writer.close()

#####  13. Interpreto topicos
topics = lda.print_topics(num_words=8)
for topic in topics:
    print(topic)
```