UNIVERSIDAD DEL QUINDÍO

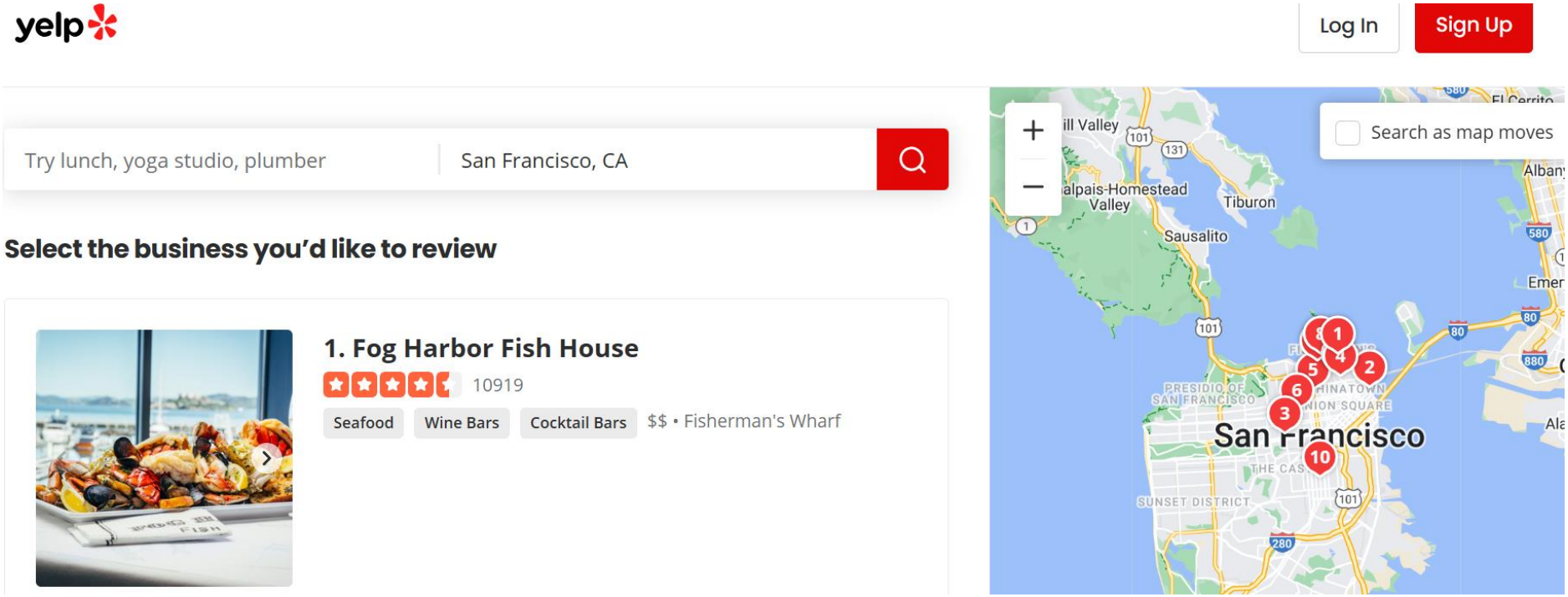
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN

Información general	
NOMBRE DE LA ASIGNATURA:	Introducción al Big Data
FECHA:	12 de septiembre de 2024
DURACIÓN ESTIMADA	4 horas
DOCENTE:	Luis Fernando Castro, PhD.
GUÍA NO.	P_1
Nombre de la guía:	Proyecto 1

Procedimiento:

PRIMERA PARTE (50 %)

Preprocesamiento de datos – Caso de estudio YELP reviews



En este caso, comprenderá qué es el procesamiento del lenguaje natural (PLN) y cómo puede resultar útil. Conoceremos NLTK, una biblioteca de Python que implementa muchos algoritmos de PNL comunes. Obtendrá información sobre los desafíos específicos de la PNL y herramientas como vectorización, palabras vacías, tokenización y etiquetado de partes del discurso para abordar estos desafíos.

Introducción

En su rol de consultor empresarial para pequeñas y medianas empresas con un gran número de clientes. Ejemplos de tales negocios podrían incluir un restaurante de comida rápida, una tienda de ropa o un distribuidor en línea de equipos para pasatiempos. Ha sido contratado para ayudar a las empresas a comprender qué influyen en las experiencias positivas y negativas de los clientes. Los clientes a menudo no están dispuestos a dar comentarios directos, pero dejan un gran número de reseñas en línea en sitios web como Yelp, Amazon, etc.

El proyecto consiste en desarrollar un servicio que permita a las empresas obtener rápidamente resúmenes útiles de sus reseñas en dichos sitios web. Un servicio de este tipo permitiría a sus clientes responder preguntas como: "¿Cuáles son los factores más importantes que generan críticas negativas?" o "¿Un cambio de política reciente mejoró nuestras revisiones?"

Procesamiento lenguaje natural

Algunos de los casos de éxito más famosos relacionados con la PNL provienen de Google, donde se utiliza para dar muy buenas respuestas a búsquedas en Internet vagas o mal escritas. Además, traducciones automáticas bastante comprensibles de texto sin formato y famosos subtítulos generados automáticamente para la mayoría de los vídeos de YouTube.

Pre procesamiento y estandarización

La estandarización del texto implica muchos pasos. Algunos de estos incluyen:

- Corrección de errores simples. Por ejemplo, un texto diferente puede utilizar codificaciones diferentes y es posible que descubra que los caracteres especiales están dañados y deben corregirse.
- Crear características (por ejemplo, etiquetar sustantivos y verbos en una oración).
- Reemplazar palabras y oraciones por completo (por ejemplo, estandarizar la ortografía cambiando "yuuuuuuck!" por "yuck", o pasos más extremos como reemplazar palabras con sinónimos)

En un sentido amplio, la estandarización permite reemplazar datos con datos más convencionales; estamos corrigiendo errores, eliminando valores atípicos y transformando características. Sin embargo, los detalles de la PNL tienden a ser más complicados. Usaremos la biblioteca Natural Language Toolkit (nltk) de Python. Esta biblioteca tiene funciones que realizan la mayoría de los conceptos básicos de PNL.

NLTK es un excelente lenguaje para aprender sobre PNL en Python. Implementa casi todos los algoritmos estándar utilizados en PNL en Python puro y es muy legible. Tiene una excelente documentación y un libro complementario y, a menudo, implementa varias alternativas al mismo algoritmo para que puedan compararse.

Cada una de las revisiones de YELP posee las siguientes características:

1. **review_id:** a unique id for that review
2. **user_id:** an anonymized identifier for the user that left the review (some users leave multiple reviews)
3. **business_id:** an anonymized identifier for the business that the review is about (most businesses have multiple reviews)
4. **stars:** the start rating (1-5) that the reviewer rated the business
5. **date:** the date the review was left
6. **text:** the full text of the review
7. **useful:** the number of readers who rated the review as useful
8. **funny:** the number of readers who rated the review as funny
9. **cool:** the number of readers who rated the review as cool

En este caso nos cencentramos en el campo 'text' que contiene el texto en lenguaje natural.

```
import nltk # imports the natural language toolkit
nltk.download('punkt')
nltk.download('stopwords')
import pandas as pd
import numpy  as np
import string
import plotly
from nltk.stem import PorterStemmer

# LOADING THE DATASET AND SEEING THE DETAILS
# If your computer can handle the entire dataset remove the nrows=5000
data = pd.read_csv('sdata.csv', nrows=5000)
data.head()
```

```
AllReviews = data['text']
AllReviews.head()
```

Tokenización de frases

Para realizer esta tarea usamos la función nltk.sent_tokenize()

En este caso un "document" se refiere a cada revisión en la colección de revisiones. La definición de documento se refiere a un item que contiene texto en lenguaje natural que es parte de una colección más grande de dichos ítems.

Exercise 1

Give an example of a question we might be able to answer with this sort of data, and another question that we'd need additional data to answer. Assume for now that all of the reviews are coming from one business.

Answer.

Aquí, cada "documento" es sólo una revisión. Echemos un vistazo a la primera revisión de "documento" en nuestro conjunto de datos y tokenícela:

```
# Print text of first Yelp review
AllReviews[0]
```

```
# sentence tokenization
sentences = nltk.sent_tokenize(AllReviews[0])
for sentence in sentences:
    print(sentence)
    print()
```

It may seem like sentence tokenization is easy, but remember that the period . can be used in many different ways. In the document:

> Tom wrote a letter to Mr. Plod, his uncle. "I am arriving on Mon. 5 Jan. Please meet me at approx. 5 p.m."

A sentence tokenizer has to be intelligent enough to tokenize this as follows:

[

"Tom wrote a letter to Mr. Plod, his uncle.",

"I am arriving on Mon. 5 Jan."

"Please meet me at approx. 5 p.m."

]

.

Tokenización de palabras

Luego de dividir los documentos en oraciones se dividen en palabras individuales.

Para ello se usa la función nltk.word_tokenize() :

```
sentences = nltk.sent_tokenize(data['text'][1])
for sentence in sentences:
    words = nltk.word_tokenize(sentence)
    print(sentence)
    print(words)
    print()
```

Exercise 2

Conduct an exploratory analysis of the sizes of reviews: find the shortest and longest reviews, then plot a histogram showing the distribution of review lengths.

Answer.

Text visualization with word clouds

Just like visualization is crucial for standard CSV data, it is also important for text data. But text doesn't lend itself to histograms or scatterplots the way that numerical or even categorical data do. In such cases, **word clouds** are a common and sometimes useful tool.

```
# Importing the required parameter for plotting
import matplotlib.pyplot as plt
from wordcloud import WordCloud
word_cloud_text = ''.join(data.text)
wordcloud = WordCloud(max_font_size=100, max_words=100, background_color="white",\
                          scale = 10,width=800, height=400).generate(word_cloud_text)
plt.figure()
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.show()
```

Exercise 3

Write a function word_cloud_rating(data, star_value) that constructs a word cloud from the subset of data that exhibit a certain star_value. Visualize the results of this function for 1-star reviews.

Answer.

Exercise 4

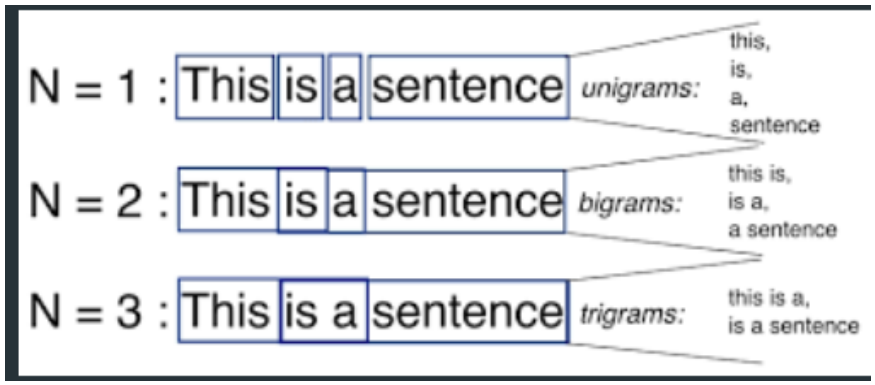
The word "good" seems to appear quite frequently in the negative reviews. Investigate why that is and come up with a reasonable explanation.

Answer.

---

n-grams

Dado que 1-gramas son insuficientes para comprender el significado de ciertas palabras en nuestro texto, es natural considerar bloques de palabras o n-gramas.



Esto ofrece ventajas y desvenajas:

- Esto conserva la estructura del documento general, y
- Allana el camino para analizar palabras en contexto; sin embargo,
- La dimensión es mucho mayor.

En la práctica, este último desafío puede ser verdaderamente abrumador. Por ejemplo, Guerra y paz tiene 3 millones de caracteres, lo que se traduce en varios cientos de miles de 1-gramas (palabras). Si consideramos que el conjunto de todos los bigramas posibles puede ser tan grande como el cuadrado de la cantidad de 1-gramas, ¡esto nos lleva a cien mil millones de bigramas posibles! Si las técnicas clásicas de ML no son adecuadas para el entrenamiento con 3 millones de caracteres, ¿cómo podrían manejar cien mil millones de dimensiones?

Por esta razón, a menudo es prudente comenzar extrayendo la mayor cantidad posible de valor de los 1-gramas, antes de avanzar hacia estructuras más complejas.

En esta sección también comenzamos a analizar nuevamente nuestra aplicación principal: calcular algunas características "interesantes" de nuestro corpus de revisiones.

Al pensar en el análisis de palabras, el tema principal de interés es encontrar una representación eficiente y de baja dimensión para facilitar la visualización de documentos y los análisis a mayor escala. Analizamos una común representación de palabras:

1. Count-based representations: word-word and word-document matrices.

Count-based representations

```
>>> from sklearn.feature_extraction.text import CountVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
>>> vectorizer = CountVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> vectorizer.get_feature_names_out()
array(['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third',
       'this'], ...)
>>> print(X.toarray())
[[0 1 1 1 0 0 1 0 1]
 [0 2 0 1 0 1 1 0 1]
 [1 0 0 1 1 0 1 1 1]
 [0 1 1 1 0 0 1 0 1]]
```

Let's create a word-document co-occurrence matrix for our set of reviews:

```
# The following code creates a word-document matrix.
from sklearn.feature_extraction.text import CountVectorizer

vec = CountVectorizer()
X = vec.fit_transform(AllReviews)
df = pd.DataFrame(X.toarray(), columns=vec.get_feature_names_out())
df.head()
```

Exercise 5

Find all the high-frequency (top 1%) and low-frequency (bottom 1%) words in the reviews overall. (Hint: import the Counter() function from the collections class.)

Answer.

Exercise 6

Write a function called top_k_ngrams(word_tokens, n, k) for printing out the top k n-grams. Use this function to get the top 10 1-grams, 2-grams, and 3-grams from the first 1000 reviews in our dataset.
Answer.

Stop words

You may have noticed a pattern in the types of words that show up in the top 10 1-grams, 2-grams, and 3-grams. In particular, these are common words that appear in every sentence of the English language: pronouns like "I", prepositions like "but", "of", "and", articles like "the", etc. These very common words are usually uninformative, and their very large occurrence values can distort the results of many NLP algorithms.

For this reason, it is common to pre-process text by removing words that you have a reason to believe are uninformative; these words are called **stop words**. Usually, it suffices to simply treat extremely common words as stop words. However, for specific types of applications it might make sense to use other stop words; e.g. the word "burger" when analyzing reviews of burger chains.

(Note that stop words are often removed by default as a cleaning step in all NLP tasks. However, sometimes they can be useful. For example in authorship attribution (automatically detecting who wrote a specific piece of text by their 'writing style'), stop words can be one of the most useful features, as they appear in nearly all texts, and yet each author uses them in slightly different ways.)

The nltk library has a standard list of stopwords, which you can download by writing nltk.download("stopwords"). We can then load the stopwords package from the nltk.corpus and use it to load the stop words:

```
nltk.download('stopwords')
from nltk.corpus import stopwords
print(stopwords.words("english"))
```

You can get a list of all the Spanish stop words as well:

```
print(stopwords.words("spanish"))
```

Exercise 7

7.1

Filter out all of the stop words in the first review of the Yelp review data and print out your answer. Additionally, print out (separately) the stopwords you found in this review.

Answer.

7.2

Modify the function top_k_ngrams(word_tokens, n, k) to remove stop words before determining the top n-grams.

Answer.

In [ ]:

```
# Removing the most basic stop words from the ntlk corpus and including only those
# words with character size above 2 so as to remove punctuations
# Use it with  n=3 , k =10
```

In some contexts, it is common to remove both very common and very *uncommon* words. The idea is that common words like "a" are almost never informative, while uncommon words like "syzygy" occur so infrequently in a corpus that many algorithms have a hard time processing them in a meaningful way. We will not deal with uncommon words today, but you should be aware that doing so improves the performance of several NLP techniques.

Encontrando las palabras principales

Hasta este punto, nos hemos centrado en las técnicas para transformar nuestros datos. Ahora estamos listos para comenzar a buscar algunas respuestas, así que tomemos un descanso de la discusión de técnicas para que podamos explorar nuestro conjunto de datos y varias formas de resumirlo.

Comenzamos por observar las palabras y los n-gramas que son más comunes en las reseñas positivas y negativas.

```
# Following code grabbed from:
# https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a
```

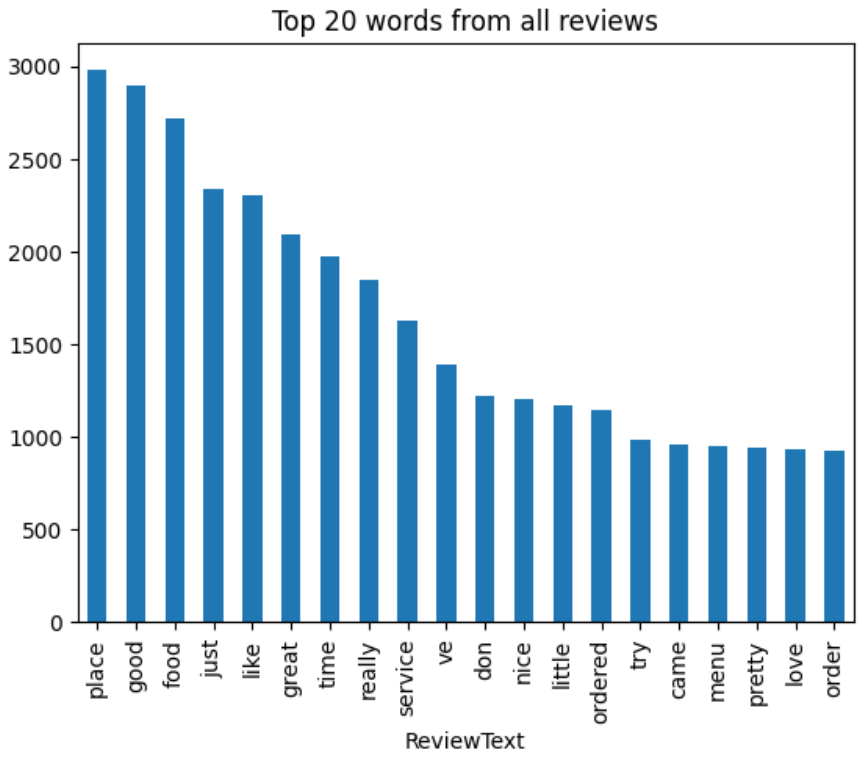
```
# we will use it in our context to create some visualizations.
def get_top_n_words(corpus, n=1,k=1):
    vec = CountVectorizer(ngram_range=(k,k),stop_words = 'english').fit(corpus)
    bag_of_words = vec.transform(corpus)
    sum_words = bag_of_words.sum(axis=0)
    words_freq = [(word, sum_words[0, idx]) for word, idx in vec.vocabulary_.items()]
    words_freq =sorted(words_freq, key = lambda x: x[1], reverse=True)
    return words_freq[:n]
```

from

```
# We start by getting a list of the most common words.

common_words = get_top_n_words(data['text'], 20,1)
for word, freq in common_words:
    print(word, freq)
df = pd.DataFrame(common_words, columns = ['ReviewText' , 'count'])
df.groupby('ReviewText').sum()['count'].sort_values(ascending=False).plot(
    kind='bar',  title='Top 20 words from all reviews')
```

from



Exercise 8

8.1

Divide the data into "good reviews" (i.e. stars rating was greater than 3) and "bad reviews" (i.e. stars rating was less or equal than 3) and make a bar plot of the top 20 words in each case. Are these results different from above?

Answer.

Well, that was pretty useless. The "good" words are mostly a mix of generic words like "place" and overtly positive words like "good" itself.

The problem here is that we are dealing with single words, which cannot convey much information out of context. The natural solution then is to deal with n-grams, so that we can get context-aware results like "good burger" or "good service" (in the positive reviews) or, as we saw, "good 45 minutes" (in the negative reviews).

8.2

Use the get_top_n_words() function to find the top 20 bigrams and trigrams (In both, bad and good reviews). Do the results seem useful?

Answer.

Question:

Look at the 5 most important bigrams for bad reviews. What *single, specific* problem seems to be the most important driver of bad reviews?

Three of the top 5 bigrams were "20 minutes", "15 minutes", and "10 minutes." These are all times, *strongly* suggesting that *waiting time for service* is a main driver for bad review scores.

Exercise 9

9.1 (Only text)

You may have noticed that many of the important "bad" bigrams included the words "like" or "just" but didn't seem very informative (e.g. "felt like", "food just"). Give some ideas of how to use this sort of observation in future pre-processing of reviews, based on the pre-processing ideas we have already studied.

Answer.

9.2 (Only text)

Building on the previous question, we note that most of the most important complaints and compliments can't be *completely* observed by looking at bigrams or trigrams. This can often be fixed by small modifications. Do the following:

1. Write down a complaint that is unlikely to be (completely) picked up by bigram analysis. Hint: what might you write if your hamburger was served cold?
2. Write down a processing step that would fix this problem. Try to find a solution that would work for several similar problems without additional human input.

Answer.

SEGUNDA PARTE (50 %)

RETO

Teniendo en cuenta la siguiente data compuesta por información de videos de YOU TUBE: MXvideos.csv

https://www.kaggle.com/datasets/datasnaek/youtube-new?resource=download&select=MXvideos.csv

Basado en lo aprendido hasta el momento y en otros conocimientos que por su parte investigue para complementar lo aprendido:

1. Aplique las técnicas que considere pertinentes para este reto

2. Teniendo en cuenta la columna 'title’ del dataset

3. Use el método Tdfidf para luego calcular la similitud entre un título de video que usted desee y los demás títulos

4. Imprima los 10 títulos de videos más recomendados (con mayor similitud)

Refs

https://github.com/jatroyano/notebook-nltk-similitud/blob/master/nltk-similitud.ipynb