Introducción al procesamiento del lenguaje natural

Obligatorio 1

2013

Grupo 01

Santiago Castro (CI: 4501276)
Jennifer Esteche (CI: 5068197)
Romina Romero (CI: 4656771)

Índice de contenido

1 Decisiones de diseño

1.1 Expresión regular

Se utilizó la siguiente expresión regular para tokenizar, cumpliendo con las restricciones

1. reconocer tokens de por lo menos 3 caracteres de largo
2. no reconocer símbolos de puntuación
3. no reconocer tokens únicamente numéricos:

([^\W\d_][^\W_]{2,})|([^\W_]+[^\W\d_][^\W_]+)|([^\W_]{2,}[^\W\d_])

donde *\W* representa un caracter que no es \w, o sea no perteneciente a [a-zA-Z0-9_], *\d* uno numérico, + que ocurre una o más veces y *{2,}* que ocurre 2 o más veces.

De esta manera, se aceptan todos los caracteres alfabéticos Unicode, incluidos aquellos con acentos.

Las 3 opciones en la expresión regular modelan los casos de que la letra obligatoria (porque no pueden ser únicamente numéricos) esté al principio, por el medio o el final.

1.2 Palabras más representativas del corpus

Se busca realizar un diccionario de las palabras más representativas de las oraciones del corpus.

1.2.1 CountVectorizer

Parámetros del constructor de CountVectorizer

input : string {‘filename’, ‘file’, ‘content’}

Este parámetro indica el tipo de entrada que se recibe (lista de nombres de archivo, nombre de un archivo o una secuencia de strings o bytes que debe analizarse directamente).

En este caso corresponde 'content', que se establece por defecto, ya que la entrada es una secuencia de strings que están cargadas en una variable.

encoding : string, ‘utf-8’ by default.

Si hay que analizar bytes o archivos, se indica el encoding utilizado para decodificar.

Se indicó que el encoding utilizado es 'unicode'.

decode_error : {‘strict’, ‘ignore’, ‘replace’}

Indica qué debe hacerse si la secuencia de bytes analizada contiene problemas de codificación, según lo indicado en *encoding*.

Se eligió el valor 'strict' para este parámetro, que es establecido por defecto, para poder

localizar errores de codificación, de existir. Éstos no deberían suceder igualmente.

strip_accents : {‘ascii’, ‘unicode’, None}

Se establece si se remueven los acentos durante el paso de preprocesamiento.

Se probó no indicar este parámetro (tomando el valor por default 'None'), y se decidió setear el valor en 'unicode', para poder reconocer como iguales las palabras cuando aparecen bien y mal escritas respecto a los acentos.

analyzer : string, {‘word’, ‘char’, ‘char_wb’} or callable

Indica si la herramienta va a analizar por palabras o utilizando n-gramas.

Se eligió utilizar el parámetro 'word' para analizar por palabras, dejado por defecto.

preprocessor : callable or None (default)

Esto es para proveer una propia función de preprocesamiento (transformación de string), preservando la de tokenización y los pasos de generación de n-gramas.

No se especificó el parámetro, ya que el valor por defecto, 'None', era el adecuado, ya que no se desea preprocesar.

tokenizer : callable or None (default)

Para proveer una propia función de tokenización de string preservando el preprocesamiento y los pasos de generación de n-gramas.

Se eligió el parámetro *tokenizer.tokenize* para aprovechar el tokenizador ya realizado con la expresión regular anteriormente.

ngram_range : tuple (min_n, max_n)

Indica los límites inferior y superior del rango de n-values para diferentes los n-gramas que serán extraídos. Todos los valores de n deben cumplir min_n <= n <= max_n.

Se eligió el valor (1,1), ya que se quiere ver cada palabra en forma independiente (se usan palabras, y no caracteres, por el valor elegido en *analyzer*). Por ejemplo, si se usara (1, 2), el vocabulario estaría compuesto por las palabras que se ingresen, más por cada bigrama que aparezca formado por dichas palabras.

stop_words : string {‘english’}, list, or None (default)

En el caso de una lista, se especifica una lista de palabras que no se desean considerar en los tokens resultantes.

Si se pasa 'english', se usa una lista predefinida para el idioma Inglés.

Se eligieron las palabras a filtrar antes de considerar las más representativas del corpus. Se excluyeron de nuestro corpus las palabras: *ademas, asi, aunque, con, cual, del,*

desde, despues, detras, donde, hay, las, los, muy, para, pero, por, que, sin, una, vez dado que eran las más frecuentes pero de hecho carecen de significado por sí mismas, y no caracterizan a las oraciones.

lowercase : boolean, default True

Convertir todos los caracteres a minúsculas antes de tokenizar.

Esta opción no se especificó. El valor por default, True, es el deseado, porque interesa reconocer si es la misma palabra, independientemente de sus mayúsculas.

token_pattern : string

Es una expresión regular que denota qué constituye un *token.* Se utiliza sólo si *tokenize == 'word'*.

Este parámetro no fue especificado, ya que se usó el tokenizador previamente definido.

max_df : float in range [0.0, 1.0] or int, optional, 1.0 by default

Cuando se construye el vocabulario, ignorar los términos que tengan una frecuencia estrictamente mayor a la dada. Si el número es float, el parámetro representa una porción de las oraciones, si es integer, cantidad absoluta. Este parámetro se ignora si el vocabulario no es None.

Este parámetro no se seteó ya que el valor por defecto era el deseado porque no se quiere filtrar las palabras por ocurrencias.

min_df : float in range [0.0, 1.0] or int, optional, 1 by default

Cuando se construye el vocabulario, ignorar los términos que tiene una frecuencia estrictamente inferior al valor dado. Corren las mismas consideraciones que para max_df.

Este parámetro no se seteó ya que el valor por defecto era el deseado, al igual que el anterior.

max_features : optional, None by default

Si no es 'None', construye un vocabulario que sólo considera el top max_features ordenado por frecuencia de términos. Este parámetro es ignorado si el vocabulario es None.

Se seleccionó este parámetro como None. La explicación está en la Sección 3.

vocabulary : Mapping or iterable, optional

Este parámetro no fue aplicado ya que se deseaba tomar el vocabulario desde el documento de entrada.

binary : boolean, False by default.

Si es True, todos los conteos no nulos se setean a 1.

Este parámetro no fue establecido, ya que servía el valor por defecto, False, porque interesa saber cuándo una palabra representativa aparece más de una vez en una oración.

dtype : type, optional

Tipo de la matriz retornada por fit_transform() o transform().

Este parámetro no fue especificado, el formato por defecto era adecuado.

1.2.2 Vocabulario

Para generar el vocabulario se decidió dejar todas las palabras que cumplieran con las restricciones previamente impuestas y sacar los acentos, ya que el corpus es pequeño, por lo que no había un gran gasto en eficiencia, y esto arrojaba mejores resultados al momento de comparar las oraciones.

Se quiere que estas palabras del vocabulario en general no sean determinantes, preposiciones ni conjunciones por ejemplo, ya que oraciones que se relacionen por tener el mismo determinante (como “una” o “los”), en realidad guardan poca semejanza. Tal vez sí podría ocurrir por determinantes como “cinco” o “aquellos”. Sí consideramos adecuando dejar sujetos, adverbios, adjetivos y pronombres personales (e interjecciones), ya que de lo contrario se perderían partes importantes de lo que se habla en cada oración. Hubiera sido de gran interés haber podido usar un POS tagger del español para clasificar las palabras tokenizadas. El NLTK no provee uno para este idioma, por eso se decidió no realizarlo.

2 Problemas encontrados y mejoras sugeridas

Para lograr un análisis más preciso, aún se deben mejorar aspectos relacionados a palabras mal escritas tales como “ambientación” y “hambientacion”, relaciones entre palabras “aderezar” y “adherezos” (que además está mal escrita), “agradable” y “agradaría”, o casos en donde las vocales aparecen muchas veces, como “muuuuy”, donde además se está enfatizando esa palabra.

La decisión de sacar los acentos con el parámetro del CountVectorizer también genera que las *ñ* se transformen en *n*, y esto provoca por ejemplo que palabras como *año* se transformen en *ano*. En este pequeño contexto no genera problemas mayores, pero en general habría que evaluar cuál es la ganancia de esta transformación, ya que sería más usual que una persona sustituya la *ñ* por *ni*, pero no por una *n* sola.

Se podrían emplear mecanismos de distancia de edición, tal vez ayudados de algún diccionario base, para detectar palabras en las que faltan letras, o se traspone su orden, por ejemplo “atencio” , “atencon” y “buna”, si es que estas no pertenecieran al diccionario dado.

El uso de abreviaciones como “atte” y “dto” tampoco está siendo considerado.

Las oraciones que no dejan espacio luego del punto no están siendo correctamente reconocidas, por ejemplo: “`Razon por la que quiero tener la certeza de si abren o no.Aclaro que ya estuve en vuestro local y me agradaria visitarlos nuevamente.Los saluda muy atte.Moreno Saba Domingo.Socio gerente de fabsab.s.r.l Argentina.`” se reconoce como una única oración.

Existe un gran problema potencial con las negaciones, que no está en el alcance de este trabajo. Por ejemplo “está bueno” y “no está bueno” son expresiones opuestas, que requieren de un procesamiento mayor, pero que pueden llegar a ser catalogadas como oraciones muy similares por el algoritmo, erróneamente. También intentar interpretar negaciones puede llegar a ser un problema de gran complejidad. Por ejemplo, se puede negar diciendo “se carece de buena calidad” en lugar de decir “no tiene buena calidad”. Se considera que no es necesario poner el “no” como palabra representativa, ya que oraciones pueden ser consideradas similares cuando se niegan cosas muy distintas.

El problema de las ambigüedades generadas debido a posible mal escritura también aparece, por ejemplo en el caso de “iva” que puede ser el verbo “ir” mal escrito o el impuesto IVA.

3 Observaciones y conclusiones

3.1 Observaciones y conclusiones generales

Al sacar los acentos de las palabras, mejora el hallazgo de palabras mal escritas, tales como el caso de "fue" y "fué", "decoración", "decoracion", "jamón" y "jamon". Esto hace que se puedan encontrar oraciones similares como: "`El provolone relleno con jamón crudo y rúcula dio en el clavo mientras esperamos la comida`" y "`Comimos pizza con queso de cabra, rúcula y jamon crudo, muy rica !!!`" se asemejen (similitud 0.72[1]).

Al generar el vocabulario, se probó tomar las veinte palabras más representativas, (y luego las cincuenta más representativas), pero el nivel de "acierto" en cuanto a similitudes de oraciones era malo, debido al escaso vocabulario manejado. Por ejemplo sucedía que muchas oraciones no tenían ninguna palabra representativa, y no se pueden comparar de forma correcta. También ocurría que oraciones muy distintas, que tenían una sola palabra representativa cada una, y esta era la misma, daban similitud 100%, lo cual era incorrecto. Al tomar todas las palabras que aparecían en el corpus y cumplían con las restricciones previas, el reconocimiento de similitudes mejoró sustancialmente.

Se corre un riesgo potencial de sobreajuste al tomar esta decisión, pero para el caso particular de este corpus, se concluyó que era una medida adecuada (se tiene conciencia de que evaluar con el corpus de entrenamiento no es lo más adecuado, pero esto va más allá del alcance de este laboratorio).

Se probó también dejando unigramas y bigramas, pero la mejora no fue significativa para nada (casi no se notó, además del impacto en el rendimiento que se vio deteriorado). Se piensa que si el corpus fuera más grande, los unigramas fallarían mucho más, y ahí sí serían de mucha más utilidad los bigramas, o los n-gramas en general con $n > 1$, ya que aumentan la probabilidad de significados parecidos.

3.2 Primer cálculo de similitudes

Las similitudes son buenas o aceptables en la mayoría de los casos.

En el caso de "`La panera la trajeron después de traer los platos (¡¡!!)`", "`Los platos EXQUISITOS!`" tienen una similitud de 0.68, debido a la palabra "platos", pero en realidad no son similares dichos comentarios en cuanto a su contenido, exceptuando el hecho de que hablan de platos. Si se considerara el contexto en el que aparecen las palabras se podría mejorar este aspecto.

Aparecen problemas de ambigüedad semántica como en el caso de la palabra "vino" que puede ser tanto sustantivo como verbo. Por ejemplo: "`El plato de pescado vino medio crudo, hubo que pedir que por favor lo cocinaran un poco más (y no esque me guste seco, simplemente le faltaba cocción)`" y "`La única queja sería q no tenían el plato q tenía previsto pedir, (ravioles con masa de remolacha que sale con cebollas al`

1 Los valores están redondeados, para ver las cifras más exactas dirigirse a las salidas del código.

`vino tinto y nueces), pero el risotto q lo sustituyó valió el cambo`", tienen una similitud de 0.60, y las palabras que se encuentran coincidentes son "plato", "pedir" y "vino", pero en realidad la palabra "vino" posee dos significados completamente distintos en estos contextos.

Además, palabras en las que cambia el género o número, o en los verbos, no están siendo reconocidos como iguales. Este es un importante aspecto a mejorar.

Se sabe que el análisis que se está haciendo es mayormente sintáctico, y que todos estos problemas son de carácter semántico, que están fuera del alcance de este laboratorio.

3.3 Segundo cálculo de similitudes

Al utilizar SpanishStemmer la mejora en similitudes encontradas fue sustancial. Palabras que diferían en género o número únicamente pudieron ser interpretadas como la misma, dando resultados más adecuados. Las grandes mejoras se deben a que se está considerando sólo las raíces (o una especie de ellas) de las palabras para compararlas.

Para los verbos suele tener problemas. "habla" y "hablo" suelen ser reconocidos como iguales, pero no "felicito" y "felicitaré".

Este ajuste hace que se vuelvan a calcular todas las similitudes, de manera más exacta. Por ejemplo el resultado más similar aquí `"Los felicito!"` y `"Felicitaciones!!!"` tiene similitud 1.0, cuando antes no era detectado debido a la diferencia morfológica de la derivación de "felicitar" de las palabras. Además, este resultado es mucho más similar que la mayor similitud detectada en el caso anterior: `"la decoracion excelente la atencion es buena"` y `"Atención excelente"`.

4 Bibliografía y referencias

[1] Información de CountVectorizer**.** En línea: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. Última visita: Setiembre 2013.

[2] Expresiones regulares en python**.** En línea: http://docs.python.org/2/library/re.html. Última visita: Setiembre 2013.

[3] NLTK**.** En línea: http://nltk.org/. Última visita: Setiembre 2013.

[4] Información de python**.** En línea: http://docs.python.org. Última visita: Setiembre 2013.

[5] Información de NLTK**.** En línea: **http://stackoverflow.com/questions/4867197/failed-loading-english-pickle-with-nltk-data-load**. Última visita: Setiembre 2013.

[6] Información de Unicode**.** En línea: **http://stackoverflow.com/questions/10288016/usage-of-unicode-and-encode-functions-in-python**. Última visita: Setiembre 2013.