**Modelado de tópicos**

En este cuaderno, realizaremos técnicas básicas de limpieza de datos textuales y luego modelaremos los datos limpios con el fin de derivar tópicos relevantes. Evaluaremos modelos de **asignación latente de Dirichlet (LDA**, por sus siglas en inglés) y ejecutaremos modelos de **factorización de matriz no-negativa** (NMF, por sus siglas en inglés). Finalmente, seremos capaces de interpretar los resultados de los modelos de tópicos e identificaremos el mejor modelo de tópicos para un determinado escenario. Veremos cómo los modelos de tópicos proporcionan información sobre la estructura subyacente de los documentos. Al final de este cuaderno, podrá crear modelos temáticos completamente funcionales para obtener valor **e información para su negocio.**

**INTRODUCCIÓN**

En el último capítulo, centramos la discusión en la preparación de datos para modelos donde usamos reducción de dimensionalidad y auto-codificación. Los grandes sets pueden ser problemáticos a la hora de presentar modelos debido a la multicolinealidad y la a extensa computación que requieren, y por lo tanto pueden dificultar la predicción en tiempo real. Una forma de resolver este problema es a través de reducción de dimensionalidad usando análisis de componentes principales. Del mismo modo, los codificadores automáticos buscan encontrar codificaciones óptimas. Podemos pensar en los codificadores automáticos como un medio de identificar términos de interacción de calidad para el conjunto de datos. Ahora pasaremos reducción de la dimensionalidad y observaremos algunas técnicas de modelado del mundo real.

El modelado de tópicos es una faceta del procesamiento del lenguaje natural (NLP, por sus siglas en inglés), el campo de la informática que explora el análisis sintáctico y semántico del lenguaje natural, que ganó mayor popularidad con el incremento en la disponibilidad de textos de conjuntos de datos. El procesamiento del lenguaje natural puede manejar el lenguaje en casi cualquier forma, incluido el texto, el habla, y las imágenes. El procesamiento del lenguaje natural no es la única aplicación del NLP, sino también el análisis de sentimientos, el reconocimiento de entidades y el reconocimiento de caracteres de objetos.

Diagram

Description automatically generated

Inmediatamente nos surge la pregunta ¿qué son los tópicos? Responderemos esa pregunta con un ejemplo. Habremos notado que en los días en que se llevan a cabo eventos importantes (como elecciones nacionales, desastres naturales o eventos deportivos), las publicaciones de las redes sociales tienden a centrarse en esos eventos. Las publicaciones de alguna manera reflejan los eventos del día, y lo hacen de diferentes maneras. Las publicaciones pueden tener, y tendrán, puntos de vista divergentes que pueden ser agrupados en clústeres de tópicos de alto nivel. Si tuviéramos tweets sobre la final del Mundial, los tópicos de esos tweets podrían cubrir puntos de vista divergentes, que van desde la calidad del arbitraje al comportamiento de los aficionados. En Estados Unidos, el presidente realiza un discurso anual entre mediados y finales de enero llamado Estado de la Unión. Con un número suficiente de publicaciones en las redes sociales, podríamos inferir o predecir las reacciones de alto nivel (tópicos) al discurso. Esto lo lograríamos agrupando las publicaciones usando las palabras claves contenidas en ellos. Los modelos de tópicos son importantes porque ofrecen la misma función para los datos textuales que las estadísticas clásicas para los datos numéricos. Es decir que proporcionan un resumen significativo de los datos. Volvamos al ejemplo del Estado de la Unión. Aquí trataríamos de determinar los puntos principales del discurso que resuenan o bien que son extraños a la audiencia.

**MODELOS DE TÓPICOS**

Los modelos de tópicos entran en la categoría de aprendizaje no supervisado porque, casi siempre, no se conocen de antemano los tópicos identificados. Por lo tanto, no existe un objetivo sobre el cual aplicar modelos de regresión o clasificación. En términos de aprendizaje no supervisado, los modelos de tópicos se parecen más a los algoritmos de agrupamiento, más específicamente al agrupamiento de k-medias. Recordemos que con clústers de k-medias, primero se establece el número de clústers y luego el modelo asigna cada uno de los datos a uno de los clústers predeterminados. Lo mismo ocurre generalmente con los modelos de tópicos. Seleccionamos el número de tópicos al inicio y luego el modelo aísla las palabras que forman esa cantidad de tópicos. Este es un excelente punto de partida para una descripción general de modelado de tópicos de alto nivel.

Antes que nada, verifiquemos tener instaladas y listas para usar las librerías correctas. En la siguiente tabla se enumeran las librerías necesarias y sus propósitos principales:

Table

Description automatically generated

Si no tenemos instalada alguna o todas librerías, debemos instalar los paquetes necesarios a través de la línea de comando usando **pip**; por ejemplo, **pip install langdetect**.

En el paso 3 del próximo ejercicio cubriremos la instalación de diccionarios de palabras desde

el paquete **nltk**. Los diccionarios de palabras son simplemente colecciones de palabras que son

adaptados para un uso específico. El stop words del diccionario de palabras instalado a continuación, contiene palabras comunes en Inglés que no aclaran el contexto, el significado,

o la intención. Estas palabras comunes podrían incluir *the, an,* *a* e *i*n. El word net

del diccionario de palabras proporciona asignaciones de palabras que ayudan en el proceso de lematización (explicado a continuación). Las asignaciones de palabras vinculan palabras como corre, correr y corrió como significando esencialmente lo mismo. A un alto nivel, los diccionarios de palabras proporcionan a los científicos de datos un medio para preparar datos de texto para su análisis sin tener un conocimiento profundo de la lingüística o pasar una enorme cantidad de tiempo definiendo listas de palabras o asignaciones de palabras.

EJERCICIO 7.01: CONFIGURANDO EL ENTORNO

Para verificar si el entorno está listo para el modelado de tópicos, realizaremos varios pasos. El primero de ellos consiste en cargar todas las librerías que serán necesarias en este capítulo:

1. Abra un nuevo cuaderno en Jupyter.

2. Importe las librerías requeridas:

Graphical user interface, text, application

Description automatically generated

Aunque no todos estos paquetes se utilizan para limpiar los datos sino que algunos sirven para armar el modelo; es importante importar todas las librerías requeridas de una vez. Si las librerías no fueron instaladas previamente, devolverán el siguiente error:

Graphical user interface, text

Description automatically generated with medium confidence

Si este error aparece, instale las librerías relevantes mediante la línea de comando anteriormente discutida. Una vez que la instalación sucedió satisfactoriamente, corra nuevamente el proceso de importar la librería usando **import**.

3. algunos procesos de limpieza y pre-procesado datos textuales requiere el uso de diccionario de palabras. A continuación instalaremos dos de estos diccionarios. Si la librería de **nltk** fue importada previamente, ejecute el siguiente código:

nltk.download('wordnet')

nltk.download('stopwords')

El resultado es el siguiente:

Text

Description automatically generated

4. Corra **matplotlib** y especifique inline así los plots se imprimen dentro del cuaderno:

%matplotlib inline

El cuaderno y el ambiente ya están listos para cargar los datos.

REVISIÓN DE ALTO NIVEL DE MODELADO DE TÓPICOS

Los modelos de tópicos son un enfoque de referencia para cuando analizamos grandes volúmenes de datos de texto que están potencialmente relacionados. Por “relacionados”, queremos decir que los documentos describen tópicos similares. Para ejecutar cualquier modelo de tópicos, los únicos datos necesarios son los propios documentos. No se requieren datos adicionales (meta o de otro tipo).

Para decirlo de una manera más simple, los modelos de tópicos identifican los tópicos abstractos (también conocidos como temas) en una colección de documentos (referidos como **corpus**), utilizando las palabras contenidas en los documentos. Es decir, si una oración contiene las palabras salario, empleado, y reunión, podemos asumir que esa oración trata o que su tópico es el trabajo. Cabe señalar que los documentos que componen el corpus no tienen por qué ser documentos según la definición tradicional como cartas o contratos. Un documento puede ser cualquier cosa que contenga texto, incluidos tweets, titulares de noticias o un discurso transcripto.

Los modelos de tópicos asumen que las palabras en el mismo documento están relacionadas y usan esa suposición para definir tópicos abstractos al encontrar grupos de palabras que aparecen repetidamente muy cerca. De esta forma, estos modelos son algoritmos clásicos de reconocimiento de patrones en los que los patrones detectados están formados por palabras. El tópico general del algoritmo de modelado tiene cuatro pasos principales:

1. Determinar el número de tópicos.

2. Escanear los documentos e identificar palabras o frases concurrentes.

3. Auto aprender grupos o clústers de palabras caracterizando documentos.

4. Outputs de tópicos abstractos que caracterizan el corpus como agrupaciones de palabras.

Como mencionamos en el Paso 1, es necesario seleccionar el número de tópicos antes de ajustar el modelo. Seleccionar una cantidad adecuada de tópicos puede ser complicado, pero, como es el caso con la mayoría de los modelos de aprendizaje automático, este parámetro se puede optimizar ajustando varios modelos con diferentes números de tópicos y seleccionando el mejor modelo en función de una métrica de rendimiento. Nos sumergiremos en este proceso nuevamente más tarde.

El siguiente es el flujo genérico de trabajo de modelado de tópicos:

Diagram

Description automatically generated

Es importante optimizar el parámetro de número de tópicos, ya que este parámetro puede tener un gran impacto en la coherencia del tópico. Esto se debe a que el modelo encuentra grupos de palabras que mejor se ajustan al corpus bajo la restricción de un número predefinido de tópicos. Si el número de tópicos es demasiado alto, los tópicos se vuelven inadecuadamente estrechos. Los tópicos demasiado específicos se denominan demasiado **cocinados**. Asimismo, si el número de tópicos es demasiado bajo, los tópicos se vuelven genéricos y vagos. Este tipo de tópicos se consideran **poco cocinados**. Los tópicos sobre cocinados o poco cocidos a veces se pueden solucionar disminuyendo o aumentando la cantidad de tópicos, respectivamente. En la práctica, un resultado frecuente e inevitable de los modelos de tópicos es que, con frecuencia, al menos un tópico será problemático.

Un aspecto clave de los modelos de tópicos es que no producen tópicos específicos de una palabra o una frase, sino colecciones de palabras, cada una de las cuales representa un tópico abstracto. Recordemos la oración imaginaria sobre el trabajo de antes. El modelo de tópico construido para identificar los tópicos de algún corpus hipotético al que pertenece esa oración no devolvería la palabra trabajo como tópico. En su lugar, devolvería una colección de palabras, como cheque de pago, empleado y jefe, palabras que describen el tópico y de las cuales se podría inferir el tópico de una palabra o una frase. Esto se debe a que los modelos de tópicos entienden la proximidad de las palabras, no el contexto. El modelo no tiene idea de lo que significa cheque de pago, empleado y jefe; sólo sabe que estas palabras, generalmente, siempre que aparecen, aparecen muy próximas entre sí:

Diagram

Description automatically generated

Los modelos de tópicos se pueden usar para predecir los tópicos que pertenecen a documentos no vistos, pero si vamos a hacer predicciones, es importante reconocer que los modelos de tópicos sólo conocen las palabras que se usan para entrenarlos. Es decir, si los documentos no vistos tienen palabras que no estaban en los datos de entrenamiento, el modelo no podrá procesar esas palabras incluso si se vinculan a uno de los tópico identificados en los datos de entrenamiento. Debido a este hecho, los modelos de tópicos tienden a usarse más para el análisis exploratorio y la inferencia que para la predicción.

Cada modelo de tópicos genera dos matrices. La primera matriz contiene palabras contra tópicos. Esta enumera cada palabra relacionada con cada tópico con alguna cuantificación de la relación. Dada la cantidad de palabras que considera el modelo, cada tópico sólo se describirá con una cantidad relativamente pequeña de palabras.

Las palabras se pueden asignar a un tópico o a varios tópicos con diferentes cuantificaciones. Si las palabras se asignan a uno o varios tópicos depende del algoritmo. De manera similar, la segunda matriz contiene documentos contra tópicos. Esta asigna cada documento a cada tópico mediante alguna cuantificación de la relación de cada combinación de tópico del documento.

Cuando se analiza el modelado de tópicos, es importante reforzar continuamente el hecho de que los grupos de palabras que representan los tópicos no están relacionados conceptualmente; están relacionados solo por proximidad. La proximidad frecuente de ciertas palabras en los documentos es suficiente para definir tópicos debido a una suposición establecida anteriormente: que todas las palabras en el mismo documento están relacionadas.

Sin embargo, esta suposición puede no ser cierta o las palabras pueden ser demasiado genéricas para formar tópicos coherentes. La interpretación de tópicos abstractos implica equilibrar las características innatas de los datos de texto con las agrupaciones de palabras generadas. Los datos de texto, y el lenguaje en general, son muy variables, complejos y contextuales, lo que significa que cualquier resultado generalizado debe consumirse con cautela.

Esto no es para minimizar o invalidar los resultados del modelo. Dados documentos cuidadosamente limpios y una cantidad adecuada de tópicos, las agrupaciones de palabras, como veremos, pueden ser una buena guía sobre lo que contiene un corpus y pueden incorporarse de manera efectiva en sistemas de datos más grandes.

Ya discutimos algunas de las limitaciones de los modelos de tópicos, pero hay algunos puntos adicionales que debemos considerar. La naturaleza ruidosa de los datos de texto puede hacer que los modelos de tópico asignen palabras no relacionadas con uno de los tópicos a ese tópico en particular.

Nuevamente, consideremos la oración sobre el trabajo de antes. La palabra reunión podría aparecer en la agrupación de palabras que representa el tópico de trabajo. También es posible que la palabra larga pueda estar en ese grupo, pero la palabra larga no está directamente relacionada con el trabajo. Larga puede estar en el grupo porque aparece con frecuencia muy cerca de la palabra reunión. Por lo tanto, larga probablemente se consideraría falsamente (o espuriamente) correlacionado con el trabajo y probablemente debería eliminarse de la agrupación de tópicos, si es posible. Las palabras falsamente correlacionadas en grupos de palabras pueden causar problemas significativos cuando analizando los datos.

Esto no es necesariamente una falla en el modelo. En cambio, es una característica que, dados datos ruidosos, el modelo podría extraer peculiaridades de los datos que podrían afectar negativamente los resultados. Las correlaciones espurias podrían ser el resultado de cómo, dónde o cuándo se recopilaron los datos. Si los documentos se recopilaron sólo en una región geográfica específica, las palabras asociadas con esa región podrían vincularse incorrectamente, aunque accidentalmente, a una o varias de las agrupaciones de palabras resultantes del modelo.

Tenga en cuenta que, con palabras adicionales en el grupo de palabras, podríamos adjuntar más documentos a ese tópico de los que deberían adjuntarse. Si reducimos la cantidad de palabras que pertenecen a un tópico, ese tópico se asignará a menos documentos. Tenga en cuenta que esto no es algo malo. Queremos que cada grupo de palabras contenga solo palabras que tengan sentido para que podamos asignar los tópicos apropiados a los documentos apropiados.

Hay muchos algoritmos de modelado de tópicos, pero quizás los dos más conocidos son Latent **Dirichlet Allocation (LDA**) y **Non-Negative Matrix Factorization (NMF**). Hablaremos de ambos en detalle más adelante.

APLICACIÓN EN LOS NEGOCIOS

A pesar de sus limitaciones, el modelado de tópicos puede proporcionar información procesable que genera valor comercial si se usa correctamente y en el contexto adecuado. A continuación revisaremos algunas de las mayores aplicaciones de los modelos de tópicos.

Uno de los casos de uso es el análisis exploratorio de datos en nuevos datos de texto donde se desconoce la estructura subyacente del conjunto de datos. Esto es el equivalente a graficar y calcular estadísticas de resumen para un conjunto de datos no visto que presenta variables numéricas y categóricas cuyas características deben comprenderse antes de que se puedan realizar análisis más sofisticados de manera razonable. Con los resultados del modelado de tópicos, la usabilidad de este conjunto de datos en futuros ejercicios de modelado es comprobable. Por ejemplo, si el modelo de tópicos arroja tópicos claros y distintos, entonces ese conjunto de datos sería un gran candidato para análisis de tipo de clústers.

La determinación de tópicos crea una variable adicional que se puede usar para ordenar, categorizar y/o fragmentar datos. Si nuestro modelo de tópicos devuelve los automóviles, la agricultura y la electrónica como tópicos abstractos, podríamos filtrar nuestro conjunto de datos de texto grande a solo los documentos con la agricultura como tópico. Una vez filtrados, podríamos realizar más análisis, incluido el análisis de sentimientos, otra ronda de modelado de tópicos o cualquier otro análisis que se nos ocurra. Más allá de definir los tópicos presentes en un corpus, el modelado de tópicos arroja mucha otra información indirectamente que podría usarse para desglosar aún más un gran conjunto de tópicos y comprender sus características.

Entre esas características está la prevalencia del tópico. Pensemos en realizar un análisis en una encuesta de respuesta abierta diseñada para medir la respuesta a un producto. Podríamos imaginar el modelo de tópicos devolviendo tópicos en forma de sentimiento. Un grupo de palabras puede ser bueno, excelente, recomendado, y de calidad; mientras que el otro puede ser basura, roto, pobre y decepcionante.

Dado este estilo de encuesta, los tópicos en sí mismos pueden no ser tan sorprendentes, pero lo que sería interesante es que podríamos contar la cantidad de documentos que contienen cada tópico y obtener información útil de ellos. A partir de los conteos, podríamos decir cosas como que el x por ciento de los encuestados tuvo una reacción positiva al producto, mientras que solo el y por ciento de los encuestados tuvo una reacción negativa. Esencialmente, lo que habríamos creado es una versión aproximada de un análisis de sentimiento.

Actualmente, el uso más frecuente de un modelo de tópicos es como componente de un motor de recomendaciones. Hoy en día, el énfasis está en la personalización: entregar productos a los consumidores que estén específicamente diseñados y seleccionados para esas personas. Pensemos sitios web, de noticias o de otro tipo, dedicados a la propagación de artículos. Empresas como Yahoo y Medium necesitan que los clientes sigan leyendo para mantenerse en el negocio, y una forma de mantener a los clientes leyendo es proporcionarles artículos que estarían más inclinados a leer. Aquí es donde entra en juego el modelado de tópico. Usando un corpus compuesto por artículos leídos previamente por un individuo, un modelo de tópicos, esencialmente, nos diría qué tipos de artículos le gusta leer a dicho suscriptor. Luego, la empresa podría ir a su inventario y encontrar artículos con tópicos similares y enviárselos a la persona a través de la página de su cuenta o correo electrónico. Esta es una curación personalizada para facilitar la simplicidad y la facilidad de uso al mismo tiempo que se mantiene el compromiso.

Antes de comenzar a preparar los datos para nuestro modelo, carguemos y exploremos rápidamente los datos.

EJERCICIO 7.02: CARGADO DE DATOS

En este ejercicio, cargaremos los datos y los formatearemos. Ejecutaremos este ejercicio en el mismo cuaderno que ejecutamos en el Ejercicio 7.01. Es muy importante comprender lo más a fondo posible el conjunto de datos con el que vamos a trabajar. Ese proceso de comprensión comienza con saber cómo se ven los datos en un nivel alto, qué tan grandes son los datos, qué columnas están presentes e identificar qué aspectos del conjunto de datos podrían ser útiles para resolver el problema que se nos ha encomendado. Respondemos a estas preguntas básicas a continuación.

Graphical user interface, text

Description automatically generated

Este es el único archivo que necesitamos para este ejercicio. Una vez que lo bajamos y guardamos en nuestro computador, los datos pueden ser cargados en el cuaderno.

1. Definir el camino hacia los datos y cargar usando **pandas:**

path = "News\_Final.csv"

df = pandas.read\_csv(path, header=0)

Nota: agregar el archivo a la misma carpeta donde tenemos abierto el cuaderno.

2. Examinar los datos brevemente ejecutando el siguiente código:

def dataframe\_quick\_look(df, nrows):

print("SHAPE:\n{shape}\n".format(shape=df.shape))

print("COLUMN NAMES:\n{names}\n".format(names=df.columns))

print("HEAD:\n{head}\n".format(head=df.head(nrows)))

dataframe\_quick\_look(df, nrows=2)

Esta función definida por el usuario devuelve la forma de los datos (el número de filas y columnas), los nombres de las columnas y las dos primeras filas de los datos:

Text, letter

Description automatically generated

3. Observar que una de las columnas, denominada **Tópicos**, en realidad contiene la información que cualquier modelo de tópicos intentaría determinar. Mire brevemente los datos del tópico provistos, para que cuando finalmente genere sus propios tópicos, los resultados se puedan comparar directamente. Ejecute la siguiente línea para imprimir los valores de tópicos únicos y su número de ocurrencias:

print("TOPICS:\n{topics}\n".format(topics=df["Topic"]\

.value\_counts()))

Obtendremos el siguiente resultado:

Text

Description automatically generated with medium confidence

4. Ahora, extraer los datos del título y transformar los datos extraídos en un objeto de lista. Imprimir los primeros cinco elementos de la lista y la longitud de la lista para confirmar que la extracción fue exitosa:

raw = df["Headline"].tolist()

print("HEADLINES:\n{lines}\n".format(lines=raw[:5]))

print("LENGTH:\n{length}\n".format(length=len(raw)))

Obtendremos el siguiente resultado:

Text, letter

Description automatically generated

Al tener los datos cargados y formateados correctamente, podemos hablar de la limpieza de datos textuales y luego de limpieza y pre-procesamiento reales. Una vez que establezcamos el proceso y lo hayamos probado en el título de ejemplo, ejecutaremos el proceso en cada título.

LIMPIEZA DE DATOS DE TEXTO

Un componente clave de todos los ejercicios de modelado exitosos es un conjunto de datos limpio que ha sido pre-procesado adecuada y suficientemente para el tipo de datos y análisis específicos que se están realizando. Los datos de texto no son una excepción, ya que son prácticamente inutilizables en su forma original. No importa qué algoritmo se esté ejecutando: si los datos no se preparan correctamente, los resultados, en el mejor de los casos, no tendrán sentido y, en el peor de los casos, serán engañosos. Como dice el dicho, basura entra, basura sale. Para el modelado de tópicos, el objetivo de la limpieza de datos es aislar las palabras en cada documento que podrían ser relevantes eliminando todo lo que podría ser obstructivo.

La limpieza y el pre-procesamiento de datos casi siempre son específicos del conjunto de datos, lo que significa que cada conjunto de datos requerirá un conjunto único de pasos de limpieza y pre-procesamiento seleccionados para manejar específicamente los problemas que contiene. Con datos de texto, los pasos de limpieza y pre-procesamiento pueden incluir filtrado de idioma, eliminación de URL y nombres de pantalla, lematización y eliminación de palabras, entre otros. Veremos esto en detalle en las próximas secciones e implementaremos estas ideas en los próximos ejercicios, donde se limpiará un conjunto de datos con titulares de noticias para el modelado de tópicos.

TÉCNICA DE LIMPIEZA DE DATOS

Recordemos que el objetivo de la limpieza de texto para el modelado de tópicos es aislar las palabras en cada documento que podrían ser relevantes para encontrar los tópicos abstractos del corpus. Esto significa eliminar palabras comunes, palabras cortas (generalmente más comunes), números y puntuación. No existe un proceso duro y rápido para limpiar datos, por lo que es importante comprender los puntos problemáticos típicos en el tipo de datos que se limpian y realizar un trabajo exploratorio extenso.

Analicemos algunas de las técnicas de limpieza de datos de texto que emplearemos. Antes de realizar cualquier tarea de modelado que involucre texto debemos determinar el(los) idioma(s) del texto. En este conjunto de datos, la mayoría de los titulares están en inglés, por lo que eliminaremos los títulos que no estén en inglés para simplificar. La construcción de modelos en datos de texto que no están en inglés requiere conjuntos de habilidades adicionales, la menor de las cuales es la fluidez en el idioma que se está modelando.

El siguiente paso crucial en la limpieza de datos es eliminar todos los elementos de los documentos que no sean relevantes para los modelos basados en palabras o que sean fuentes potenciales de ruido que podrían oscurecer los resultados. Los elementos que deben eliminarse pueden incluir direcciones de sitios web, signos de puntuación, números y palabras vacías. Las palabras vacías son básicamente palabras simples y de uso común (incluidos nosotros, somos y el/la). Es importante señalar que no existe un diccionario definitivo de palabras vacías; en cambio, cada diccionario varía ligeramente. A pesar de las diferencias, cada diccionario contiene una cantidad de palabras comunes que se supone que son independientes del tópico. Los modelos de tópicos intentan identificar palabras que son lo suficientemente frecuentes e infrecuentes para ser descriptivas de un tópico abstracto.

La eliminación de direcciones de sitios web tiene una motivación similar. Las direcciones de sitios web específicos aparecerán muy raramente, pero incluso si una dirección de sitio web específica apareciera lo suficiente como para vincularla a un tópico, las direcciones de sitios web no se pueden interpretar de la misma manera que las palabras. La eliminación de información irrelevante de los documentos reduce la cantidad de ruido que podría impedir la convergencia del modelo u oscurecer los resultados.

La **lematización**, como la detección del lenguaje, es un componente importante de todas las actividades de modelado que involucran texto. Es el proceso de reducir las palabras a su forma básica como una forma de agrupar palabras que deberían ser todas iguales pero no lo son debido a varios cambios en el tiempo o la parte del discurso. Considere las palabras corriendo, corre y corrió. Estas tres palabras tienen la forma base de correr. Un gran aspecto de la lematización es que analiza todas las palabras de una oración , es decir, considera el contexto antes de determinar cómo alterar cada palabra. La lematización, como la mayoría de las técnicas de limpieza anteriores, simplemente reduce la cantidad de ruido en los datos, para que podamos identificar tópicos limpios e interpretables.

Ahora, con un conocimiento básico de las técnicas de limpieza textual, aplicaremos estas técnicas a los datos del mundo real.

EJERCISIO 7.03: LIMPIEZA DE DATOS PASO A PASO

En este ejercicio, aprenderemos cómo implementar algunas técnicas clave para limpiar datos de texto. Cada técnica se explicará a medida que avancemos en el ejercicio. Después de cada paso de limpieza, el título del ejemplo se imprime mediante **print**, por lo que podremos ver la evolución desde los datos sin procesar hasta los datos listos para el modelo:

1. Seleccionar el sexto título como ejemplo sobre el cual construiremos y probaremos el proceso de limpieza. El sexto titular no es una elección aleatoria; lo elegimos porque contiene problemas específicos abordaremos durante el proceso de limpieza:

example = raw[5]

print(example)

Obtendremos el siguiente resultado:



2. Utilizar la librería **langdetect** para detectar el idioma de cada titular. Si el idioma no es inglés (**en**), elimine ese título del conjunto de datos. La función **detect** simplemente detecta el idioma del texto que se le pasa. Cuando la función no detecta un idioma, lo que hace periódicamente, simplemente establezca el idioma en **none** para eliminarlo más adelante.

def do\_language\_identifying(txt):

try: the\_language = langdetect.detect(txt)

except: the\_language = 'none'

return the\_language

print("DETECTED LANGUAGE:\n{lang}\n"\

.format(lang=do\_language\_identifying(example)))

Obtendremos el siguiente resultado:

A picture containing text

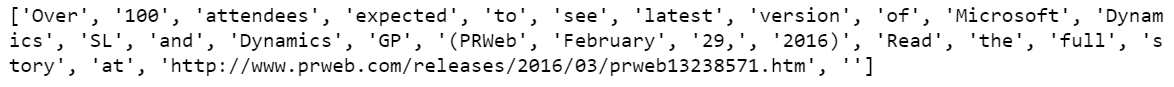
Description automatically generated

3. Dividir la cadena que contiene el título en partes, llamadas **tokens**, utilizar los espacios en blanco. El objeto devuelto es una lista de palabras y números que componen el título. Dividir la cadena de titulares en tokens simplifica el proceso de limpieza y pre-procesamiento. Hay varios tipos de tokenizadores disponibles. Tener en cuenta que NLTK en sí proporciona varios tipos de tokenizadores. Cada uno de los tokenizadores considera diferentes formas de dividir la oración en tokens. El más simple es dividir el texto en espacios en blanco.

example = example.split(" ")

print(example)

Obtendremos el siguiente resultado:



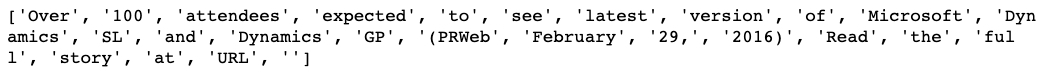
4. identificar todos los URLs mediante una búsqueda de expresiones regulares para tokens que contengan **http://** o **https://.** Reemplazar las URL con la cadena 'URL':

example = ['URL' if bool(regex.search("http[s]?://", i)) \

else i for i in example]

print(example)

Obtendremos el siguiente resultado:

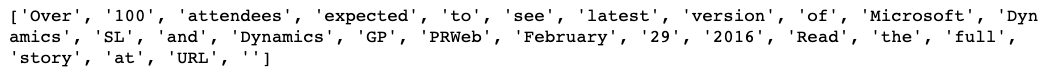


5. Reemplazar todos los símbolos de puntuación y nueva línea (**\n**) con cadenas vacías usando expresiones regulares:

example = [regex.sub("[^\\w\\s]|\n", "", i) for i in example]

print(example)

Obtendremos el siguiente resultado:

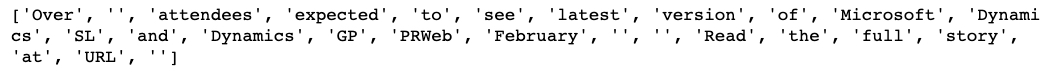


6. Remplazar todos los números con cadenas vacías usando expresiones regulares:

example = [regex.sub("^[0-9]\*$", "", i) for i in example]

print(example)

Obtendremos el siguiente resultado:

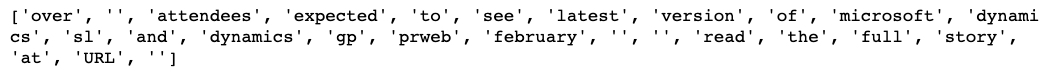


7. Cambiar todas las letras mayúsculas a minúsculas. Convertir todo a minúsculas no es un paso obligatorio, pero ayuda a reducir la complejidad. Con todo en minúsculas, hay menos de lo que hacer un seguimiento y, por lo tanto, menos posibilidades de error:

example = [i.lower() if i not in ["URL"] else i for i in example]

print(example)

Obtendremos el siguiente resultado:



8. Eliminar la cadena 'URL' que se agregó como marcador de posición en el Paso 4. La cadena 'URL' agregada anteriormente no es realmente necesaria para el modelado. Si parece inofensivo dejarlo, considere que la cadena 'URL' podría aparecer de forma natural en un título y no queremos aumentar artificialmente su número de apariciones. Además, la cadena 'URL' no aparece en todos los titulares, por lo que, al dejarla, podríamos estar creando sin querer una conexión entre las cadenas 'URL' y un tópico:

example = [i for i in example if i not in ["URL",""]]

print(example)

Obtendremos el siguiente resultado:



9. Cargue en el diccionario de palabras vacías de **nltk** e imprímalo:

list\_stop\_words = nltk.corpus.stopwords.words("english")

list\_stop\_words = [regex.sub("[^\\w\\s]", "", i) \

for i in list\_stop\_words]

print(list\_stop\_words)

Obtendremos el siguiente resultado:

A close-up of a document

Description automatically generated with medium confidence

Antes de usar el diccionario, es importante reformatear las palabras para que coincidan con el formato de nuestros titulares. Eso implica confirmar que todo está en minúsculas y sin puntuación.

10. Luego de formatear correctamente el diccionario de palabras irrelevantes, lo podemos utilizar para eliminar todas las palabras vacías del título:

example = [i for i in example if i not in list\_stop\_words]

print(example)

Obtendremos el siguiente resultado:



11. Realizar la lematización definiendo una función que se pueda aplicar a cada titular individualmente. La lematización requiere que se cargue el diccionario **wordnet**. La función **morphy** toma cada palabra individual en un texto y devuelve su forma estándar si la reconoce. Por ejemplo, si la entrada de palabra se está ejecutando o se ejecutó, la función morphy devolvería ejecutar:

def do\_lemmatizing(wrd):

out = nltk.corpus.wordnet.morphy(wrd)

return (wrd if out is None else out)

example = [do\_lemmatizing(i) for i in example]

print(example)

Obtendremos el siguiente resultado:



12. Eliminar todas las palabras con una longitud de cuatro o menos de la lista de fichas. La suposición en torno a este paso es que las palabras cortas son, en general, más comunes y, por lo tanto, no impulsarán los tipos de conocimientos que buscamos extraer de los modelos de tópicos. Tener en cuenta que eliminar palabras de cierta longitud no es una técnica que deba usarse todo el tiempo; es solo para casos específicos. Por ejemplo, las palabras cortas a veces pueden ser muy indicativas de tópicos como en el caso de identificar animales (por ejemplo, perro, gato, pájaro).

example = [i for i in example if len(i) >= 5]

print(example)

Obtendremos el siguiente resultado:



Ahora que realizamos la limpieza y pre-procesamiento individualmente en un titular, necesitamos aplicar esos pasos a cada uno de los casi 100,000 titulares. La forma más eficiente de hacerlo es escribiendo una función que contenga todos los pasos descriptos anteriormente y aplicar esa función a cada documento en el corpus en alguna forma iterativa. Llevaremos a cabo ese proceso en el próximo ejercicio.

EJERCISIO 7.04: LIMPIEZA DE DATOS COMPLETA

En este ejercicio, consolidaremos los pasos 2 a 12 del ejercicio 7.03 en una función que podemos aplicar a cada titular. La función tomará un título en formato de cadena como entrada y la salida será un título limpio como una lista de fichas. Los modelos de tópicos requieren que los documentos tengan el formato de cadenas en lugar de como listas de tokens, por lo que en el Paso 4, las listas de tokens se vuelven a convertir en cadenas:

1. Definir una función que contenga todos los pasos individuales del proceso de limpieza

del Ejercicio 7.03:

def do\_headline\_cleaning(txt):

# identify language of tweet

# return null if language not English

lg = do\_language\_identifying(txt)

if lg != 'en':

return None

# split the string on whitespace

out = txt.split(" ")

# identify urls

# replace with URL

out = ['URL' if bool(regex.search("http[s]?://", i)) \

else i for i in out]

# remove all punctuation

out = [regex.sub("[^\\w\\s]|\n", "", i) for i in out]

# remove all numerics

out = [regex.sub("^[0-9]\*$", "", i) for i in out]

2. Ejecutar la función en cada titular. La función de **map** en Python es una buena manera

para aplicar una función definida por el usuario a cada elemento de una lista. Convertir el objeto del **map** a una lista y asígnela a la variable **clean**. La variable **clean** es una lista de listas:

tick = time()

clean = list(map(do\_headline\_cleaning, raw))

print(time()-tick)

3. En **do\_headline\_cleaning**, obtenemos **None** si el idioma del título se detecta como cualquier otro idioma que no sea inglés. Los elementos de la lista limpia final solo deben ser listas, no **None**, por lo tanto, debemos eliminar todos los tipos **None**. Usar **print** para mostrar los primeros cinco titulares limpios y la longitud de la variable limpia:

clean = list(filter(None.\_\_ne\_\_, clean))

print("HEADLINES:\n{lines}\n".format(lines=clean[:5]))

print("LENGTH:\n{length}\n".format(length=len(clean)))

Obtendremos el siguiente resultado:

Text

Description automatically generated

4. Para cada título individual, concatenar los tokens usando un espacio en blanco

separador. Los titulares ahora deberían ser una colección no estructurada de palabras,

sin sentido para el lector humano, pero ideal para el modelado de tópicos:

clean\_sentences = [" ".join(i) for i in clean]

print(clean\_sentences[0:10])

Los titulares limpios deberían re-ensamblar lo siguiente:

A close-up of a document

Description automatically generated with low confidence

En resumen, lo que el trabajo de limpieza y pre-procesamiento hace efectivamente es eliminar el ruido de los datos para que el modelo pueda perfeccionar los elementos de los datos que podrían en realidad impulsar los conocimientos. Por ejemplo, las palabras que son independientes de cualquier tópico no deben ser tópicos informativos, pero solo por accidente, si se deja, podría serlo. En un esfuerzo por evitar lo que podríamos llamar una señal falsa, eliminamos esas palabras. Igualmente, dado que los modelos de tópicos no pueden discernir el contexto, la puntuación es irrelevante y, por lo tanto, es remoto. Incluso si el modelo pudiera encontrar los tópicos sin eliminar el ruido de los datos, los datos sin limpiar podrían tener miles o millones de palabras adicionales y caracteres aleatorios para analizar (dependiendo de la cantidad de documentos en el corpus), lo que podría aumentar significativamente las demandas computacionales. Entonces, la limpieza de datos es una parte integral del modelado de tópicos. En la siguiente actividad lo pondremos en práctica.

ACTIVIDAD 7.01: CARGADO Y LIMPIEZA DE DATOS DE TWITTER

En esta actividad, cargaremos y limpiaremos los datos de Twitter para realizar el modelado en

actividades posteriores. Continuaremos con el uso de los datos de titulares, así que completaremos esta actividad en un cuaderno Jupyter separado, pero con todos los mismos requisitos y librerías importadas.

El objetivo es tomar los datos sin procesar del tweet, limpiarlos y producir el mismo resultado que hicimos en el paso 4 del ejercicio anterior. La salida debe ser una lista cuya longitud es

similar al número de filas en el archivo de datos sin procesar, pero potencialmente no igual. Esto se debe a que los tweets pueden perderse en el proceso de limpieza por muchas razones, como por ejemplo que el tweet está escrito en un idioma que no es inglés. Cada elemento de la lista debe representar un tweet y debe contener solo las palabras en el tweet que podrían

ser relevante para la formación del tópico.

Los pasos para completar la actividad son los siguientes:

1. Importar las librerías necesarias.

2. Cargar los datos de LA Times health Twitter (**latimeshealth.txt**) desde https://packt.live/2Xje5xF.

3. Correr un análisis exploratorio rápido para evaluar el tamaño de los datos y estructura.

4. Extraer los datos del tweet y convertirlo en una lista de objetos.

5. Escribir una función para realizar detección de idioma y tokenización en los espacios en blanco. Luego reemplazar los nombres de pantalla y URL con **SCREENNAME** y **URL**, respectivamente. La función también debe eliminar la puntuación, los números y los reemplazos de **SCREENNAME** y **URL**. Convertir todo a minúsculas, excepto **SCREENNAM**E y **URL**. Debe eliminar todas las palabras vacías, realizar lematización, y mantener palabras con cinco o más letras solamente.

6. Aplicar la función descripta en el Paso 5 a cada tweet.

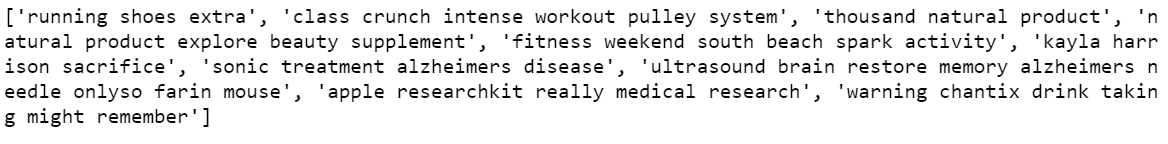
7. Remover los elementos de las líneas de salida que son igual a **None**.

8. Convertir los elementos de cada tweet nuevamente en una cadena. Concatenar usando

espacio en blanco.

9. Mantener el cuaderno abierto para futuras actividades.

Obtendremos el siguiente resultado:



ALOCACIÓN LATENTE DE DIRICHLET

En 2003, David Blei, Andrew Ng, y Michael Jordan publicaron un artículo sobre el algoritmo del modelado de tópicos conocido como **Latent Dirichlet Allocation (LDA).** LDA es un

modelo probabilístico generativo. Esto significa que el proceso de modelado comienza con el texto y funciona hacia atrás a través del proceso que suponemos que lo generó, con el fin de

identificar los parámetros de interés. En este caso, son los tópicos que generaron los datos que son de interés. Este proceso es la forma más básica de LDA, pero para aprendizaje, también es el más comprensible.

Hay M documentos para modelado de tópicos disponibles dentro del corpus. Cada documento puede ser considerado como la secuencia de N palabras, por ejemplo, la sequencia w1, w2, …, wn.

Para cada documento del corpus, el proceso generativo asumido es:

1. Seleccionar , donde N es el número de palabras y es el parámetro que controla distribución de Poisson.

2. Seleccionar *)* , donde está la distribución de tópicos.

3. Para cada N palabras, , seleccionar el tópico seleccionar la palabra desde .

Veamos el proceso generativo con un poco más de detalle. Los tres paso anteriores se repiten para cada documento en el corpus. El primer paso es elegir el número de palabras en el documento tomando muestras, en la mayoría de los casos, de la distribución de Poisson. Es importante notar que, debido a que N es independiente de las otras variables, la aleatoriedad asociada con su generación se ignora en su mayoría en el derivación del algoritmo.

Luego deseleccionar N, debemos generar la mezcla de tópicos o distribución de tópicos, únicos para cada documento. Pensemos en esto como una lista de tópicos por documento con

probabilidades de que representan el monto del documento representado por cada tópico. Consideremos tres tópicos: A, B y C. Un ejemplo podría ser 100% del tópico A, 75% del tópico B y 25% del tópico C, o una infinidad de otras combinaciones.

Por último, las palabras específicas en el documento se seleccionarán a través de una declaración de probabilidad condicionado al tópico seleccionado y la distribución de palabras para ese tópico. Tener en cuenta que los documentos no se generan realmente de esta manera, pero es un proxy razonable.

Este proceso puede considerarse como una distribución sobre distribuciones. Un documento es

seleccionado de la colección (distribución) de documentos, y se selecciona un tópico (a través de la distribución multinomial) de la distribución de probabilidad de tópicos para ese documento, generado por la distribución de Dirichlet.

Diagram

Description automatically generated

La forma más sencilla de construir la fórmula que representa la solución LDA es a través de una representación gráfica. Esta particular representación se denomina modelo gráfico de notación de placa ya que utiliza placas para representar los dos pasos iterativos del proceso.

Recordemos que el proceso generativo se ejecutó para cada documento del corpus, por lo que la placa más externa (etiquetada como *M*) representa la iteración sobre cada documento. De manera similar, la iteración sobre palabras en el Paso 3 está representada por la placa más interna del diagrama, etiquetada como *N*.

Los círculos representan los parámetros, las distribuciones y los resultados. El círculo etiquetado como *W* es la palabra seleccionada, que es el único dato conocido y, como tal, se utiliza para realizar ingeniería inversa en el proceso generativo. Además de *W*, las otras cuatro variables en el diagrama se definen de la siguiente manera:

α: hiperparámetro del documento del tópico para la distribución de Dirichlet.

β: distribución de palabras para cada tópico.

Z: variable latente del tópico.

θ: variable latente para la distribución de tópicos de cada documento.

α β controlan la frecuencia de tópicos de los documentos y la frecuencia de palabra en los tópicos. Si α aumenta, los documentos se vuelven cada vez más similares a medida que aumenta el número de tópicos en cada documento. Por otro lado, si α disminuye, los documentos se vuelven cada vez más disímiles a medida que disminuye el número de tópicos en cada documento. El parámetro β se comporta de manera similar. Si β aumenta, se usan más palabras del documento para modelar un tópico, mientras que un valor más bajo hace que se use una cantidad menor de palabras para un tópico. Dada la complejidad de las distribuciones en LDA, no existe una solución directa, por lo que se requiere algún tipo de algoritmo de aproximación para generar los resultados. Analizaremos luego el algoritmo de aproximación estándar para LDA.

INFERENCIA VARIACIONAL

El gran problema con LDA es que la evaluación de las probabilidades condicionales (las distribuciones) es inmanejable, por lo que en lugar de calcularlas directamente, las probabilidades se aproximan. La inferencia variacional es uno de los algoritmos de aproximación más simples, pero tiene una derivación extensa que requiere un conocimiento significativo de la probabilidad. Para dedicar más tiempo a la aplicación de LDA, veremos en este cuaderno algunos detalles de alto nivel sobre cómo se aplica la inferencia variacional en este contexto, pero no exploraremos completamente el algoritmo.

Tomemos un momento para trabajar intuitivamente con el algoritmo de inferencia variacional. Empecemos asignando aleatoriamente cada palabra de cada documento del corpus a uno de los tópicos. Luego, para cada documento y cada palabra en cada documento por separado, calculemos dos proporciones. Esas proporciones serían la proporción de palabras en el documento que está actualmente asignado al tópico, *P(Tópico|Documento)* y la proporción de asignaciones en todos los documentos de una palabra específica al tópico, *P(Palabra|Tópico)*. Multipliquemos las dos proporciones y usemos la proporción resultante para asignar la palabra a un nuevo tópico. Repitamos este proceso hasta que se alcance un estado estable en el que las asignaciones de tópicos no cambien significativamente. Estas asignaciones luego se utilizan para estimar la mezcla de tópicos dentro del documento y la mezcla de palabras dentro del tópico.

Diagram

Description automatically generated

El pensamiento detrás de la inferencia variacional es que, si la distribución real es intratable, entonces se debe encontrar una distribución más simple, llamémosla distribución variacional, muy cercana a la distribución verdadera, que es manejable, para que la inferencia sea posible. En otras palabras, dado que es imposible inferir la distribución real debido a la complejidad de la distribución real, intentaremos encontrar una distribución más simple que sea una excelente aproximación de la distribución real.

Veamos un ejemplo. La inferencia variacional es como tratar de ver animales en un zoológico lleno de gente. Los animales del zoológico están en un hábitat cerrado que, en este ejemplo, es la distribución posterior. Los visitantes en realidad no pueden ingresar al hábitat, por lo que los visitantes deben conformarse con ver el hábitat desde la posición más cercana posible, que es la aproximación posterior (es decir, la mejor aproximación del hábitat). Si hay mucha gente en el zoológico, puede ser difícil llegar a ese punto de vista óptimo. La gente generalmente comienza en la parte de atrás de la multitud. y avanza estratégicamente hacia ese punto de vista óptimo. El paso de los visitantes desde la parte trasera de la multitud al punto de vista óptimo, es el camino de optimización. La inferencia variacional es simplemente el proceso de acercarse lo mejor posible al punto deseado sabiendo que en realidad no se puede alcanzar el punto deseado.

Para empezar, seleccionaremos una familia de distribuciones *q* (es decir, binomial, gaussiana, exponencial, etc.) condicionada a nuevos parámetros variacionales. Los parámetros están optimizados para que la distribución original (la distribución posterior para aquellos familiarizados con las estadísticas Bayesianas) y la distribución variacional sean lo más cercanas

posible. La distribución variacional será lo suficientemente cercana a la distribución posterior original para ser utilizada como proxy, haciendo que cualquier inferencia hecha sobre ella sea aplicable a la distribución posterior original. La fórmula genérica de la familia de distribuciones, *q*, es la siguiente:

Hay una gran colección de distribuciones variacionales potenciales que se pueden usar como una aproximación para la distribución posterior. Se selecciona una distribución variacional inicial de la colección, que actúa como punto de partida para un proceso de optimización que iterativamente se acerca más y más a la distribución óptima. Los parámetros óptimos son los parámetros de la distribución que mejor se aproximan al posterior. La similitud de las dos distribuciones se mide utilizando la divergencia de **Kullback-Leibler (KL)**. La divergencia KL representa la cantidad esperada de error generado si aproximamos una distribución con otra. La distribución con parámetros óptimos tendrá la divergencia KL más pequeña cuando se mida con la distribución real.

Una vez que identificamos la distribución óptima, es decir que los parámetros óptimos fueron identificados, podemos aprovechar para producir las matrices de salida y ejecutar cualquier inferencia requerida.

BOLSA DE PALABRAS

El texto no puede pasarse directamente a un algoritmo de aprendizaje automático; primero debe codificarse numéricamente. Una forma sencilla de trabajar con texto en el aprendizaje automático es a través de un modelo de bolsa de palabras, que elimina toda la información sobre el orden de las palabras y se centra estrictamente en el grado de presencia (es decir, el recuento o la frecuencia) de cada palabra.

Podemos usar la librerís **sklearn** de Python para transformar el vector limpio creado en el ejercicio anterior en la estructura que requiere el modelo LDA. Dado que LDA es un modelo probabilístico, no queremos escalar ni ponderar las ocurrencias de palabras. En su lugar, sólo ingresaremos los recuentos sin procesar.

La lista de cadenas limpias que obtuvimos en el Ejercicio 7.04 será ;a entrada al modelo de bolsa de palabras. El resultado será el número del documento, la palabra como su codificación numérica, y un recuento del número de veces que esa palabra aparece en ese documento. Estos tres elementos se presentarán como una tupla y un número entero.

La tupla sería algo así como (0, 325), donde 0 es el número de documento y 325 es la palabra codificada numéricamente. Tengamos en cuenta que 325 será la codificación de esa palabra en todos los documentos. El entero sería entonces el conteo. Los modelos de bolsa de palabras que ejecutaremos en este cuaderno son de **sklearn** y se llaman **CountVectorizer** y **TfIdfVectorizer**. El primer modelo devuelve los recuentos sin procesar y el segundo devuelve un valor escalado, del que hablaremos un poco más adelante.

Es importante saber que los resultados de ambos modelos temáticos que tratamos en este cuaderno pueden variar de una ejecución a otra, incluso cuando los datos son los mismos, debido a la aleatoriedad. Ni las probabilidades en LDA ni los algoritmos de optimización son deterministas, así que no debemos sorprendernos si los resultados difieren ligeramente de los resultados que se muestran a continuación. En el siguiente ejercicio, ejecutaremos el vectorizador de conteo para codificar numéricamente nuestros documentos, de modo que podamos continuar con el modelado de tópicos usando LDA.

EJERCICIO 7.05: CREACIÓN DE MODELO DE BOLSA DE DATOS USANDO COUNT VECTORIZER

En este ejercicio, ejecutaremos **CountVectorizer** en **sklearn** para convertir el vector limpio de titulares que creamos previamente en una estructura de datos de bolsa de palabras. Además, definiremos algunas variables que se utilizarán a lo largo del proceso de modelado:

1. Definir **number\_words, number\_docs y number\_features**. Las dos primeras variables controlan la visualización de los resultados de LDA. La variable **number\_features** controla la cantidad de palabras que se mantendrán en el espacio:

number\_words = 10

number\_docs = 10

number\_features = 1000

2. Ejecutar el vectorizador de conteo e imprimir la salida. Hay tres entradas cruciales,

que son **max\_df, min\_df y max\_features**. Estos parámetros filtran el número de palabras en el corpus a aquellas que pueden influir en el modelo.

Las palabras que sólo aparecen en una pequeña cantidad de documentos son demasiado raras para ser atribuibles a cualquier tópico, por lo que se utiliza **min\_df** para descartar palabras que aparecen menos veces que el número especificado de documentos. Palabras que aparecen en demasiados documentos no son lo suficientemente específicas como para estar vinculadas a tópicos específicos, por lo que **max\_df** se utiliza para desechar palabras que aparecen en mayor porcentaje que el especificado en los documento. Por último, no queremos sobre ajustar el modelo, por lo que el número de palabras utilizadas para ajustar el modelo se limitará al número especificado que ocurre con mayor frecuencia (**max\_features**) de las palabras:

vectorizer1 = sklearn.feature\_extraction.text\

.CountVectorizer(analyzer="word",\

max\_df=0.5,\

min\_df=20,\

max\_features=number\_features)

clean\_vec1 = vectorizer1.fit\_transform(clean\_sentences)

print(clean\_vec1[0])

Obtendremos el siguiente resultado:

Text

Description automatically generated

3. Extraer los nombres de las características y las palabras del vectorizador. El modelo sólo recibe las codificaciones numéricas de las palabras, por lo que la fusión del vector de nombres con los resultados facilitará la interpretación:

feature\_names\_vec1 = vectorizer1.get\_feature\_names()

En este ejercicio realizamos la enumeración de los documentos para usar en el modelo LDA. El formato requerido es una bolsa de palabras. Es decir, un modelo de bolsa de palabras es simplemente una lista de todas las palabras que aparecen en cada documento con un recuento del número de veces que aparece cada palabra en cada documento específico. Una vez realizada esta tarea con **sklearn**, es hora de explorar el proceso de evaluación de modelos LDA.

PERPLEJIDAD

Los modelos generalmente tienen métricas que se pueden aprovechar para evaluar su desempeño. Los modelos de tópicos no son diferentes, aunque el rendimiento, en este caso, tiene una definición ligeramente diferente. En la regresión y la clasificación, los valores pronosticados se pueden comparar con los valores reales a partir de los cuales podemos calcular medidas claras de rendimiento.

Con los modelos de tópicos, la predicción es menos confiable, porque el modelo solo conoce las

palabras con las que se entrenó y los documentos nuevos no pueden contener ninguna de esas palabras, a pesar de presentar los mismos tópicos. Debido a esa diferencia, los modelos de tópicos se evalúan usando una métrica específica para los modelos de lenguaje, llamada **perplejidad**.

La perplejidad, abreviado como PP, mide el número de diferentes igualmente la mayoría

palabras probables que pueden seguir a cualquier palabra dada en promedio. Consideremos dos palabras como ejemplo: el/la y anunciar. La palabra el/la puede referirse con alta probabilidad a un gran número de palabras. Por otro lado, la palabra anunciar aunque puede seguir refiriéndose a un gran número de palabras, será en menos medida que el/la.

La idea es que palabras que, en promedio, puedan ir seguidas de un menor número de

Igualmente más probables, son más específicas y pueden estar más estrechamente vinculadas a los tópicos. Como tal, puntuaciones más bajas de perplejidad implican mejores modelos de lenguaje. La perplejidad es muy similar a la entropía, pero la perplejidad se usa típicamente porque es más fácil de interpretar. Se puede utilizar para seleccionar el número óptimo de tópicoss. Con *m* siendo el número de palabras en la secuencia de palabras, la perplejidad se define como:

En esta fórmula, son las palabras que componen algún documento en el conjunto de datos de la prueba. La probabilidad conjunta de esas palabras, P (es una medida de qué tan bien el documento de prueba encaja en el modelo existente. Probabilidades más altas sugieren una modelos más fuertes. La probabilidad se eleva a la potencia para normalizar la puntuación por el número de palabras en cada documento y para hacer que los valores más bajos sean óptimos. En ambos cosos, estos cambios aumentan la interpretabilidad del puntaje. La puntuación de perplejidad, como la raíz del el error cuadrático medio, no es muy significativo como una métrica independiente. Tiende a utilizarse como métrica de comparación. Es decir, se construyen varios modelos para los cuales la perplejidad de las puntuaciones se calculan y comparan para identificar el mejor modelo con el cual seguir.

Como mencionamos anteriormente, LDA tiene dos entradas requeridas. La primera son los documentos en sí, y la segundo es la cantidad de tópicos. Seleccionar un número apropiado de tópico puede ser muy complicado. Un enfoque para encontrar el número óptimo de tópico es buscar en varios números de tópico y seleccionar el número de tópico que corresponde a la menor puntuación de perplejidad. En el aprendizaje automático, este enfoque se conoce como búsqueda en grilla. En el siguiente ejercicio, pondremos a trabajar la búsqueda en grilla para encontrar el número óptimo de tópicos.

EJERCICIO 7.06: SELLECIÓN DEL NÚMERO DE TÓPICOS

En este ejercicio, usamos las puntuaciones de perplejidad para modelos LDA que se ajustan a una cantidad variable de tópicos para determinar la cantidad de tópicos con los cuales avanzar. Tengamos en cuenta que el conjunto de datos original tenía los titulares clasificados en cuatro tópicos. Veamos si este enfoque devuelve cuatro tópicos:

1. Definir una función que se ajuste a un modelo LDA en varios números de tópicos y calcular la puntuación de perplejidad. Devolver dos elementos: un DataFrame que tiene el número de tópicos con su puntaje de perplejidad y el número de tópicos con el puntaje mínimo de perplejidad como un número entero:

def perplexity\_by\_ntopic(data, ntopics):

output\_dict = {"Number Of Topics": [], \

"Perplexity Score": []}

for t in ntopics:

lda = sklearn.decomposition.LatentDirichletAllocation(\

n\_components=t, \

learning\_method="online", \

random\_state=0)

lda.fit(data)

output\_dict["Number Of Topics"].append(t)

output\_dict["Perplexity Score"]\

.append(lda.perplexity(data))

output\_df = pandas.DataFrame(output\_dict)

index\_min\_perplexity = output\_df["Perplexity Score"]\

.idxmin()

output\_num\_topics = output\_df.loc[\

index\_min\_perplexity, # index \

"Number Of Topics" # column

]

return (output\_df, output\_num\_topics)

2. Ejecutar la función definida en el Paso 1. La entrada de **ntopics** es una lista de números de tópicos que pueden tener cualquier longitud y contener cualquier valor. Imprimir el marco de datos:

df\_perplexity, optimal\_num\_topics = \

perplexity\_by\_ntopic(clean\_vec1, ntopics=[1, 2, 3, 4, 6, 8, 10])

print(df\_perplexity)

Obtendremos el siguiente resultado:

Table

Description automatically generated

3. Representar gráficamente las puntuaciones de perplejidad en función del número de tópicos. Esta es solo otra forma de ver los resultados contenidos en el DataFrame del Paso 2:

df\_perplexity.plot.line("Number Of Topics", "Perplexity Score")

Obtendremos el siguiente gráfico:

Chart, line chart

Description automatically generated

Como muestran el DataFrame y la trama, tres es el número óptimo de tópicos que utilizan perplejidad. Tener el número de tópicos establecido en cuatro produjo la segunda perplejidad más baja. Por lo tanto, si bien los resultados no coincidieron exactamente con la información contenida en el conjunto de datos original, los resultados son lo suficientemente cercanos como para generar confianza en el enfoque de búsqueda en grilla para identificar la cantidad óptima de tópicos. Puede haber varias razones por las que la búsqueda en grilla arrojó tres en lugar de cuatro, que analizaremos en un próximo ejercicio.

Ahora que hemos seleccionamos la cantidad óptima de tópicos, usaremos esa cantidad de tópicos para construir nuestro modelo LDA oficial. Luego, ese modelo se utilizará para crear visualizaciones y definir la lista de tópicos presentes en el corpus.

EJERCISIO 7.07: CORRIENDO LDA

En este ejercicio, implementaremos LDA y examinaremos los resultados. LDA genera dos matrices. La primera es la matriz tópico-documento y la segunda es la matriz palabra-tópico. Veremos estas matrices como devueltas por el modelo y como tablas bien formateadas que son más fáciles de interpretar:

1. Ajustear un modelo LDA utilizando la cantidad óptima de tópicos que se encuentran en el Ejercicio 7.06:

lda = sklearn.decomposition.LatentDirichletAllocation\

(n\_components=optimal\_num\_topics,\

learning\_method="online",\

random\_state=0)

lda.fit(clean\_vec1)

Obtendremos el siguiente resultado:

Text

Description automatically generated

2. Generar la matriz tópico-documento y su forma para confirmar que se alinea con la cantidad de tópicos y la cantidad de documentos. Cada fila de la matriz corresponde la distribución de tópicos por documento:

lda\_transform = lda.transform(clean\_vec1)

print(lda\_transform.shape)

print(lda\_transform)

Obtendremos el siguiente resultado:

Text

Description automatically generated

3. Generar la matriz palabra-tópico y su forma para confirmar que se alinea con la cantidad de características (palabras) especificadas en el Ejercicio 7.05 y la cantidad de tópicos ingresados. Cada fila es básicamente la prevalencia de asignaciones a ese tópico de cada palabra. La puntuación de prevalencia se puede transformar en la distribución de palabras por tópico:

lda\_components = lda.components\_

print(lda\_components.shape)

print(lda\_components)

Obtendremos el siguiente resultado:

Graphical user interface, text, application

Description automatically generated

4. Definir una función que formatee las dos matrices de salida en tablas fáciles de leer:

def get\_topics(mod, vec, names, docs, ndocs, nwords):

# word to topic matrix

W = mod.components\_

W\_norm = W / W.sum(axis=1)[:, numpy.newaxis]

# topic to document matrix

H = mod.transform(vec)

W\_dict = {}

La función puede ser complicada de navegar. Comencemos por crear las matrices *W* y *H,* lo que incluye convertir los recuentos de asignaciones de *W* en la distribución de palabras por tópico. Luego, iterar sobre los tópicos. Dentro de cada iteración, identifique las principales palabras y documentos asociados con cada tópico. Convierte los resultados en dos DataFrames.

5. Ejecutar la función definida en el Paso 4:

W\_df, H\_df = get\_topics(mod=lda, \

vec=clean\_vec1, \

names=feature\_names\_vec1, \

docs=raw, \

ndocs=number\_docs, \

nwords=number\_words)

6. Imprimir la tópico- palabra DataFrame. Aquí se muestran las 10 palabras principales (por valor de distribución) asociadas con cada tópico. A partir de este DataFrame, podemos identificar los tópicos abstractos que representan las agrupaciones de palabras.

print(W\_df)

Obtendremos el siguiente resultado:

Text

Description automatically generated

7. Imprimir el tópico-documento DataFrame. Aquí se muestran los 10 documentos con los que cada tópico está más relacionado. Los valores son de la distribución de tópicos por documento:

print(H\_df)

Obtendremos el siguiente resultado:

A screenshot of a computer

Description automatically generated with low confidence

Los resultados del DataFrame de tópico- palabra muestran que los tópicos abstractos son Barack Obama, la economía y Microsoft. Lo interesante es que la agrupación de palabras que describen la economía hacen referencias a Palestina. Los cuatro tópicos especificados en

el conjunto de datos original se representa en la salida de DataFrame de tópico-palabra, pero no de la manera totalmente diferenciada que esperábamos. Podríamos estar enfrentando uno de dos problemas.

Primero, el tópico que hace referencia tanto a la economía como a Palestina podría estar poco cocinado, lo que significa que aumentar la cantidad de tópicos puede solucionar el problema. El otro potencial problema es que LDA no maneja bien los tópicos relacionados. En el ejercicio 7.09, intentaremos ampliar el número de tópicos, lo que nos dará una mejor idea de por qué una de las agrupaciones de palabras es aparentemente una mezcla de tópicos.

VISUALIZACION

La salida de los modelos LDA en Python usando **sklearn** puede ser difícil de interpretar en forma cruda. Como es el caso en la mayoría de los ejercicios de modelado, las visualizaciones pueden ser un gran beneficio cuando se trata de interpretar y comunicar los resultados del modelo. Una librería de Python, **pyLDAvis**, se integra directamente con el objeto del modelo **sklearn** para producir gráficos sencillos. Esta herramienta de visualización devuelve un histograma que muestra las palabras más relacionadas con cada tópico y un biplot, de uso frecuente en PCA, donde cada círculo corresponde a un tópico. A partir del biplot, sabemos la prevalencia de cada tópico en todo el corpus, lo que se refleja en el área del círculo, y la similitud de los tópicos, que se refleja en la cercanía de los círculos.

El escenario ideal es que los círculos se extiendan por toda la trama y tengan un tamaño razonable y consistente. Es decir, queremos que los tópicos sean distintos y que aparezcan uniformemente en todo el corpus. Además de los gráficos de **pyLDAvis**, aprovecharemos el modelo t-SNE, discutido en un capítulo anterior, para producir una representación bidimensional de la matriz tópico-documento, una matriz donde cada fila representa un documento y cada columna representa la probabilidad de ese tópico que describe el documento.

Habiendo completado el ajuste del modelo LDA, creemos algunos gráficos para ayudarnos a profundizar en los resultados.

EJERCISIO 7.08: VISUALIZACIÓN LDA

La visualización es una herramienta útil para explorar los resultados de los modelos de tópicos. En este ejercicio, veremos tres visualizaciones diferentes. Esas visualizaciones son histogramas básicos y visualizaciones especiales que usan t-SNE y PCA:

1. Ejecutar y mostrar **pyLDAvis**. Esta trama es interactiva. Al hacer clic en cada círculo, se actualiza el histograma para mostrar las principales palabras relacionadas con ese tópico específico. La siguiente es una vista de este diagrama interactivo:

lda\_plot = pyLDAvis.sklearn\

.prepare(lda, clean\_vec1, vectorizer1, R=10)

pyLDAvis.display(lda\_plot)

Los gráficos muestran lo siguiente:

Chart, bubble chart

Description automatically generated

2. Definir una función que se ajuste a un modelo t-SNE y luego represente gráficamente los resultados. Después de definirla, se describirán en detalle las piezas de la función, para que los pasos sean claros:

def plot\_tsne(data, threshold):

# filter data according to threshold

index\_meet\_threshold = numpy.amax(data, axis=1) >= threshold

lda\_transform\_filt = data[index\_meet\_threshold]

# fit tsne model

# x-d -> 2-d, x = number of topics

tsne = sklearn.manifold.TSNE(n\_components=2, \

verbose=0, \

random\_state=0, \

angle=0.5, \

init='pca')

tsne\_fit = tsne.fit\_transform(lda\_transform\_filt)

# most probable topic for each headline

most\_prob\_topic = []

**Paso 1**: La función comienza filtrando la matriz de tópico-documento utilizando un valor de umbral de entrada. Hay decenas de miles de titulares, y cualquier trama que incorpore todos los titulares será difícil de leer y, por lo tanto, no será útil. Por lo tanto, esta función solo traza un documento si uno de los valores de distribución es mayor o igual que el valor del umbral de entrada:

index\_meet\_threshold = numpy.amax(data, axis=1) >= threshold

lda\_transform\_filt = data[index\_meet\_threshold]

**Paso 2**: Una vez filtrados los datos, ejecute t-SNE, donde el número de componentes es dos, por lo que podemos trazar los resultados en dos dimensiones:

tsne = sklearn.manifold.TSNE(n\_components=2, \

verbose=0, \

random\_state=0, \

angle=0.5, \

init='pca')

tsne\_fit = tsne.fit\_transform(lda\_transform\_filt)

**Paso 3**: Crear un vector con un indicador de qué tópico está más relacionado con cada documento. Este vector se usará para codificar con colores la trama por tópico:

most\_prob\_topic = []

for i in range(tsne\_fit.shape[0]):

most\_prob\_topic.append(lda\_transform\_filt[i].argmax())

**Paso 4**: Comprender la distribución de los tópicos en el corpus y el impacto del filtrado de umbral, la función devuelve la longitud del vector de tópico, así como los propios tópicos con la cantidad de documentos para los que ese tópico tiene el mayor valor de distribución:

print("LENGTH:\n{}\n".format(len(most\_prob\_topic)))

unique, counts = numpy.unique(numpy.array(most\_prob\_topic), \

return\_counts=True)

print("COUNTS:\n{}\n".format(numpy.asarray((unique, counts)).T))

**Paso 3**: Crear y devolver el gráfico:

color\_list = ['b', 'g', 'r', 'c', 'm', 'y', 'k']

for i in list(set(most\_prob\_topic)):

indices = [idx for idx, val in enumerate(most\_prob\_topic) \

if val == i]

matplotlib.pyplot.scatter(x=tsne\_fit[indices, 0], \

y=tsne\_fit[indices, 1], \

s=0.5, c=color\_list[i], \

label='Topic' + str(i), \

alpha=0.25)

matplotlib.pyplot.xlabel('x-tsne')

matplotlib.pyplot.ylabel('y-tsne')

matplotlib.pyplot.legend(markerscale=10)

3. Ejecutar la función

plot\_tsne(data=lda\_transform, threshold=0.75)

Obtendremos el siguiente resultado:

Chart, scatter chart

Description automatically generated

Las visualizaciones muestran que el modelo LDA con tres tópicos está produciendo buenos resultados en general. En el biplot, los círculos son de tamaño mediano, lo que sugiere que los tópicos aparecen consistentemente a lo largo del corpus y los círculos también tienen un buen espaciado. El gráfico t-SNE muestra grupos claros que respaldan la separación entre los círculos representados en el bigráfico. El único problema evidente, que se discutió anteriormente, es que uno de los tópicos tiene palabras que no parecen pertenecer a ese tópico.

EJERCICIO 7.09: INTENTANDO CUATRO TÓPICOS

En este ejercicio, LDA se ejecuta con el número de tópicos establecido en cuatro. La motivación para hacer esto es tratar de resolver lo que podría ser un tópico poco cocinado del modelo LDA de tres tópicos que tiene palabras relacionadas tanto con Palestina como con la economía. Primero ejecutaremos los pasos y luego exploraremos los resultados al final:

1. Correr el modelo LDA con el número de tópicos igual a cuatro:

lda4 = sklearn.decomposition.LatentDirichletAllocation(\

n\_components=4, # number of topics data suggests \

learning\_method="online", \

random\_state=0)

lda4.fit(clean\_vec1)

Obtendremos el siguiente resultado:

Text

Description automatically generated

2. Ejecutar la función **get\_topics** que definimos anteriormente para producir las tablas tópico- palabra y tópico-documento más legibles:

W\_df4, H\_df4 = get\_topics(mod=lda4, \

vec=clean\_vec1, \

names=feature\_names\_vec1, \

docs=raw, \

ndocs=number\_docs, \

nwords=number\_words

3. Imprimir la tabla tópico- palabra:

print(W\_df4)

Obtendremos el siguiente resultado:

A close-up of a document

Description automatically generated with medium confidence

3. Imprimir la tabla tópico- documento:

print(H\_df4)

Obtendremos el siguiente resultado

A picture containing text, newspaper

Description automatically generated

5. Mostrar los resultados de modelo LDA usando **pyLDAvis:**

lda4\_plot = pyLDAvis.sklearn\

.prepare(lda4, clean\_vec1, vectorizer1, R=10)

pyLDAvis.display(lda4\_plot)

El gráfico es el siguiente:

Chart, bubble chart

Description automatically generated

Al observar la tabla de palabras y tópicos, vemos que los cuatro tópicos encontrados por este modelo se alinean con los cuatro tópicos especificados en el conjunto de datos original. Esos tópicos son Barack Obama, Palestina, Microsoft y la economía. La pregunta ahora es, ¿por qué el modelo construido con cuatro tópicos tiene una puntuación de perplejidad más alta que el modelo con tres tópicos? Esa respuesta proviene de la visualización producida en el Paso 5.

El biplot tiene círculos de tamaño razonable, pero dos de esos círculos están bastante juntos, lo que sugiere que esos dos tópicos (Microsoft y la economía) son muy similares. En este caso, la similitud en realidad tiene sentido intuitivo. Microsoft es una importante empresa global que impacta y es impactada por la economía. El siguiente paso, si tuviéramos que hacer uno, sería ejecutar el gráfico t-SNE para verificar si los grupos en el gráfico t-SNE se superponen.

ACTIVIDAD 7.02 LDA Y TWEETTS DE SALUD

En esta actividad, aplicaremos LDA a los datos de tweets de salud cargados y limpiados en la Actividad 7.01. Recordemos que debemos usar el mismo cuaderno que usamos en esa actividad. Una vez ejecutados los pasos, discutamos los resultados del modelo. ¿Estas agrupaciones de palabras tienen sentido?

Para esta actividad, imaginemos que estamos interesados en adquirir una comprensión de alto nivel de los principales tópicos de salud pública. Es decir, de lo que se está hablando en el mundo de la salud. Hemos recopilado algunos datos que podrían arrojar luz sobre esta consulta. La forma más fácil de identificar los tópicos principales en el conjunto de datos, como hemos comentado, es el modelado de tópicos.

Estos son los pasos para completar la actividad:

1. Especificar las variables **number\_words, number\_docs, and number\_**

**features**

2. Crear un modelo de bolsa de datos y asignar los nombres a otra variable para usar más tarde.

3. Identificar el número óptimo de tópicos.

4. Ajustar el modelo LDA usando el número óptimo de tópicos.

5. Crear e imprimir la tabla de tópico-palabra.

6. Crear e imprimir la tabla de tópico-documento.

7. crear una visualización blipot.

8. mantener el cuaderno abierto para futuros modelos.

Obtendremos el siguiente resultado:

Chart

Description automatically generated

EJERCICIO 7.10: CREAR UN MODELO DE BOLSA DE PALABRAS USANDO TF-IDF

1. Correr el vectorizador TF-IDF e imprimir la salida de las primeras líneas:

vectorizer2 = sklearn.feature\_extraction.text.TfidfVectorizer\

(analyzer="word",\

max\_df=0.5, \

min\_df=20, \

max\_features=number\_features,\

smooth\_idf=False)

clean\_vec2 = vectorizer2.fit\_transform(clean\_sentences)

print(clean\_vec2[0])

Obtendremos el siguiente resultado:

Text

Description automatically generated

2. Devolver los nombres (las palabras reales en el diccionario del corpus) para usar al analizar la salida. Recordemod que hicimos lo mismo cuando ejecutamos CountVectorizer en el Ejercicio 7.05:

feature\_names\_vec2 = vectorizer2.get\_feature\_names()

feature\_names\_vec2

Obtendremos el siguiente resultado:

Text

Description automatically generated

Text, letter

Description automatically generated

En este ejercicio, resumimos el corpus en forma de un modelo de bolsa de palabras. Se calcularon los pesos para cada combinación de palabras del documento.

FACTORIZACIÓN DE MATRIZ NO-NEGATIVA

A diferencia de LDA, la factorización de matriz no negativa (**NMF**, por sus siglas en inglés) no es un modelo probabilístico. En cambio, es, como su nombre lo indica, es un enfoque que involucra álgebra lineal. Daniel D. Lee y H. Sebastian Seung introdujeron en 1999 el uso de la factorización matricial como un enfoque para el modelado de tópicos.

Las principales diferencias entre PCA y NMF son que PCA requiere que los componentes sean perpendiculares y les permite ser positivos o negativos. NMF requiere que los componentes de la matriz no sean negativos, lo que debería tener sentido si piensa en este requisito en el contexto de los datos. Los tópicos no pueden estar relacionados negativamente con los documentos, y las palabras no pueden estar relacionadas negativamente con los tópicos.

Si no nos convence, intentemos interpretar un peso negativo asociando un tópico a un documento. Sería algo así como el tópico T constituye -30% del documento D; pero ¿qué significa eso? No tiene sentido, por lo que NMF tiene requisitos no negativos para cada parte de la factorización de la matriz.

Definamos la matriz a factorizar, *X*, como una matriz término-documento donde las filas son palabras y las columnas son documentos. Cada elemento de la matriz *X* es el número de ocurrencias de la palabra i (la fila) en el documento j (la columna) o alguna otra cuantificación de la relación entre la palabra i y el documento j. La matriz, X, es naturalmente una matriz dispersa, ya que la mayoría de los elementos de la matriz término-documento serán cero, ya que cada documento solo contiene un número limitado de palabras.

Diagram

Description automatically generated with medium confidence

La factorización matricial toma la forma , donde las dos matrices de componentes, W y H, representan los tópicos como colecciones de palabras y los pesos de los tópicos para cada documento, respectivamente. Más específicamente, es una matriz de palabra por tópico, mientras que es una matriz de tópico por documento y, como se indicó anteriormente, es una matriz de palabra por documento.

Podemos pensar en esta factorización como una suma ponderada de agrupaciones de palabras que definen tópicos abstractos. El símbolo de equivalencia en la fórmula para la factorización de la matriz es un indicador de que la factorización WH es una aproximación y, por lo tanto, el producto de esas dos matrices no reproducirá exactamente la matriz original del término-documento.

El objetivo, como sucedió con LDA, es encontrar la aproximación más cercana a la matriz original. Al igual que X, tanto W como H son matrices dispersas, ya que cada tópico solo se relaciona con unas pocas palabras y cada documento es una combinación de solo una pequeña cantidad de tópicos-un tópico en muchos casos.

LA NORMA DE FROBENIUS

El objetivo de resolver NMF es el mismo que el de LDA: encontrar la mejor aproximación. Para medir la distancia entre la matriz de entrada y la aproximación, NMF podemos usar prácticamente cualquier medida de distancia, pero el estándar es la Norma de Frobenius, también conocida como Norma Euclidiana. La Norma de Frobenius es la suma de los errores cuadrados de los elementos expresados matemáticamente como .

Con la medida de distancia seleccionada, el siguiente paso es definir la función objetivo. La minimización de la Norma de Frobenius devolverá la mejor aproximación de la matriz término-documento original y, por tanto, los tópicos más razonables. Tengamos en cuenta que la función objetivo se minimiza con respecto a W y H para que ambas matrices no sean negativas. Se expresa como .

EL ALGORITMO DE ACTIALIZACIÓN MULTIPLICATIVA

El algoritmo de optimización utilizado para resolver NMF por Lee y Seung en su artículo de 1999 es el algoritmo de actualización multiplicativa, y sigue siendo una de las soluciones más utilizadas. Lo implementaremos en futuros ejercicios y actividades.

Las reglas de actualización, tanto para W como para H, se obtienen expandiendo la función objetivo y tomando las derivadas parciales con respecto a W y H. Las derivadas no son difíciles, pero requieren un conocimiento de álgebra lineal bastante amplio y requieren mucho tiempo. Por lo que saltearemos los derivados y solo indiquemos las actualizaciones. Tengamos en cuenta que, en las reglas de actualización, i es la iteración actual y T significa la transposición de la matriz. La primera regla de actualización es la siguiente:

La segunda regla de actualización es la siguiente:

W y H se actualizan iterativamente hasta que el algoritmo converge. También se puede demostrar que la función objetivo no es creciente; es decir, con cada actualización iterativa de W y H, la función objetivo se acerca al mínimo. Tengamos en cuenta que el optimizador de actualización multiplicativo, si se reorganizan las reglas de actualización, es un gradiente re-escalado del algoritmo de descenso.

El componente final para construir un algoritmo NMF exitoso es inicializar las matrices de componentes W y H para que la actualización multiplicativa funcione rápidamente. Un enfoque popular para inicializar matrices es la **Descomposición de Valores Singulares (SVD,** por sus siglas en inglés), que es una generalización de la descomposición de Eigen.

En la implementación de NMF que haremos en los próximos ejercicios, las matrices

se inicializan a través de la descomposición de valores singulares dobles no negativos, que es

básicamente una versión más avanzada de SVD que es estrictamente no negativa. Los detalles completos de estos algoritmos de inicialización no son importantes para comprender NMF. Sólo

debemos tener en cuenta que los algoritmos de inicialización se utilizan como punto de partida para la optimización de algoritmos y puede acelerar drásticamente la convergencia.

EJERCICIO 7.11: FACTORIZACION DE LA MATRIZ NO-NEGATIVA

En este ejercicio, ajustaremos el algoritmo NMF y generaremos las mismas dos tablas de resultados que hicimos anteriormente con LDA. Esas tablas son la tabla de palabras y tópicos, que muestran las 10 palabras principales asociadas con cada tópico, y la tabla de documentos y tópicos, que muestra los 10 documentos principales asociados con cada tópico.

Hay dos parámetros adicionales en la función del algoritmo NMF que no discutimos aún, que son **alpha** y **l1\_ratio**. Si un modelo sobreajustado es motivo de preocupación, estos parámetros controlan cómo (**l1\_ratio**) y el grado en que se aplica la regularización (**alpha**) a la función objetivo:

1. Definir el modelo NMF y llamar a la función **fit** usando la salida del TF-IDF vectorizer:

nmf = sklearn.decomposition.NMF(n\_components=4, \

init="nndsvda", \

solver="mu", \

beta\_loss="frobenius", \

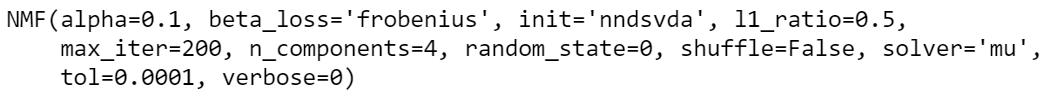
random\_state=0, \

alpha=0.1, \

l1\_ratio=0.5)

nmf.fit(clean\_vec2)

Obtendremos el siguiente resultado:



2. Correr la función **get\_topics** para obtener las dos tablas de salida:

W\_df, H\_df = get\_topics(mod=nmf, \

vec=clean\_vec2, \

names=feature\_names\_vec2, \

docs=raw, \

ndocs=number\_docs, \

nwords=number\_words)

3. Imprimir la tabla **W**:

print(W\_df)

Obtendremos el siguiente resultado:

Text

Description automatically generated

4. Imprimir la tabla H:

print (H\_df)

Obtendremos el siguiente resultado:

A picture containing text, newspaper

Description automatically generated

La tabla de palabras-tópicos contiene agrupaciones de palabras que sugieren los mismos tópicos abstractos que el modelo LDA de cuatro tópicos producido en el Ejercicio 7.09. Sin embargo, la parte interesante de la comparación es que algunas de las palabras individuales contenidas en estas agrupaciones son nuevos o están en un nuevo lugar en la agrupación. Esto no es sorprendente dado que las metodologías son distintas. Dada la alineación con los tópicos especificados en el conjunto de datos original, demostramos que ambas metodologías son herramientas efectivas para extraer la estructura temática subyacente del corpus.

EJERCICIO 7.12: VISUALIZANDO NMF

El objetivo de este ejercicio es visualizar los resultados de NMF. La visualización de los resultados da una idea de la distinción de los tópicos y la prevalencia de cada tópico en el corpus.

1. Ejecutar **transform** en los datos limpios para obtener las asignaciones de documentos de tópico. Imprimir tanto la forma como un ejemplo de los datos:

nmf\_transform = nmf.transform(clean\_vec2)

print(nmf\_transform.shape)

print(nmf\_transform)

Obtendremos el siguiente resultado:

Text

Description automatically generated

2. Correr la función **plot\_tsne** para ajustar un modelo t-SNE y graficar los resultados:

plot\_tsne(data=nmf\_transform, threshold=0)

Obtendremos el siguiente resultado:

Chart, scatter chart

Description automatically generated

El gráfico t-SNE, sin umbral especificado, muestra cierta superposición de tópicos y una clara discrepancia en la frecuencia de tópicos en todo el corpus. Estos dos hechos explican por qué, al utilizar la perplejidad, el número óptimo de tópicos fue de tres. Parece haber cierta correlación entre los tópicos que el modelo no puede acomodar por completo. Incluso con

la correlación entre tópicos el modelo está encontrando los tópicos que debería cuando el número de tópicos se establece en cuatro.

En resumen, NMF es un modelo de tópico no probabilístico que busca responder la misma pregunta que LDA está tratando de responder. Utiliza un concepto popular de álgebra lineal conocido como factorización de matrices, que es el proceso de dividir una matriz grande e intratable en matrices más pequeñas y fáciles de interpretar que se pueden aprovechar para responder muchas preguntas sobre los datos. Recordemos que el requisito no negativo no tiene sus raíces en las matemáticas, sino en los datos mismos. No tiene sentido que los componentes de cualquier documento sean negativos.

En muchos casos, NMF no funciona tan bien como LDA, porque LDA incorpora distribuciones anteriores que agregan una capa adicional de información para ayudar a informar las agrupaciones de palabras del tópico. Sin embargo, sabemos que hay casos, especialmente cuando los tópicos están altamente correlacionados, en los que NMF tiene un mejor desempeño. Uno de esos casos fue el dato titular en el que se basaron todos los ejercicios.

ACTIVIDAD 7.03: FACTORIZACIÓN NO-NEGATIVA DE LA MATRIZ

Esta actividad es la suma del análisis de modelado de tópicos realizado en los datos de salud de Twitter cargados y limpiados en la Actividad 7.01 y 7.02. La ejecución de NMF es sencilla y requiere una codificación limitada. Podemos aprovechar esta oportunidad para jugar con los parámetros del modelo mientras pensamos en las limitaciones y beneficios de NMF.

Los pasos a seguir son los siguientes:

1. Crear el modelo de bolsa de palabras apropiado y generar los nombres de las funciones como otra variable.

2. Definir y ajustar el algoritmo NMF utilizando el valor de número de tópicos **(n\_componentes**) de la Actividad 7.02.

3. Obtener las tablas tópico-documento y palabra-tópico. Explorar las agrupaciones de palabras e intente definir los tópicos abstractos. ¿Puedes cuantificar los significados de las agrupaciones de palabras? ¿Tienen sentido las agrupaciones de palabras? ¿Son los resultados similares a los producidos usando LDA?

4. Ajustar los parámetros del modelo y correr nuevamente los Pasos 3 y 4. Cómo cambiaron los resultados?

Obtendremos el siguiente resultado:

Text

Description automatically generated

RESUMEN

Cuando debemos extraer información de una gran colección de documentos aún no vista, el modelado de tópicos es un gran enfoque, ya que proporciona información sobre la estructura subyacente de los documentos. Es decir, los modelos de tópicos encuentran agrupaciones de palabras utilizando la proximidad, no el contexto.

En este cuaderno, aprendimos cómo aplicar dos de los algoritmos de modelado de tópicos más comunes y efectivos: la asignación de Dirichlet latente y la factorización de matriz no negativa. Ahora deberíamos sentirnos cómodos limpiando documentos de texto sin formato utilizando varias técnicas diferentes; técnicas que se pueden utilizar en muchos otros escenarios de modelado. Continuaremos aprendiendo cómo convertir el corpus limpio en la estructura de datos adecuada de recuentos de palabras sin procesar o pesos de palabras por documento mediante la aplicación de modelos de bolsa de palabras.

El enfoque principal del cuaderno fue ajustar los dos modelos de tópicos, incluida la optimización de la cantidad de tópicos, la conversión de la salida en tablas fáciles de interpretar y la visualización de los resultados. Con esta información, deberíamos poder aplicar modelos de tópicos completamente funcionales para obtener valor e información para cualquier negocio.