LDA VS LSA

Minería de datos

Alumno:

Isaac Perez Amayo

Leonardo Daniel Rosas Garcia

Kevin Raymond Hernandez

Programa educativo:

Ingeniería en Datos e Inteligencia Organizacional.

Presentado a:

Jose Antonio Leon Borges

Cancún, Quintana Roo Marzo del 2022

Índice:

Introducción

En este proyecto haremos una comparación de dos algoritmos de modelado de tópicos, usaremos latent semantic indexing y latent dirichlet allocation, pero primero qué es el Topic modeling?, el topic modeling consiste en descubrir automáticamente los temas de los documentos,estos son algoritmos de análisis de texto no supervisado que se utiliza para encontrar el grupo de palabras del documento dado. nosotros trabajaremos con una colección de documentos que habla sobre los tópicos de machine learning.

Problemática:

En este proyecto resolveremos cual de los dos algoritmos del modelado de tópicos es el más óptimo para el análisis de grandes volúmenes de textos, realizaremos la implementación de ambos algoritmos ya dichos que son LDA y LSA, las problemáticas que tenemos en la realización de este proyecto es que la base de datos con la que debemos trabajar, es texto no pre-procesado así que tenemos que trabajar en pre-procesando la información de manera que todas las palabras de los textos estén limpios para poder utilizar ese texto en los algoritmos de LSA y LDA, también presentamos la problemática que en los algoritmos de minería de texto para poder realizar las métricas solo tenemos dos posibles métricas ya que no hay muchas que podamos usar.

Objetivos:

Nuestros objetivos de este proyecto es determinar qué algoritmo es el más óptimo para determinar la selección de tópicos.

Justificación:
La justificación de este proyecto es que al analizar el conjunto de documentos por sus tópicos podemos identificar automáticamente los tópicos más latentes en el área del machine learning, estos algoritmos nos ayudan mucho dado que lo que logran es un gran ahorro de tiempo y permiten generar más tópicos de los temas más populares, al igual, con esto se logra que los artículos y documentos de machine learning en vez de leer temas que ya no son relevantes pues no tomarlos en cuenta y dedicarse a lo más demandado.

Marco teórico:

LDA : Latent Dirichlet Allocation o conocido como LDA es un modelo generativo que permite que conjuntos puedan ser explicados por grupos no observados que explican porqué algunas partes de los datos son similares.LDA tiene una mezcla de varias características y este usa la distribución categórica tiene una distribución a priori de dirichlet. La característica principal es que LDA sigue la hipótesis de bolsa de palabras es decir que no importa el orden de palabras esta característica es la característica que reconoce la frecuencia en cada palabra y permite que sean intercambiables y permita mayores métodos matemáticos.

LSA: el algoritmo Latent Semantic Analysis también conocido como Latent Semantic Index, LSA se caracteriza por ser un algoritmo que utiliza el modelo Bag of Words, de esta manera se obtiene una matriz que muestra la ocurrencia de los términos en el documento, donde las filas representan términos y las columnas representan los documentos, lo que permite el

modelaje de tópicos es que LSA aprende temas latentes realizando una descomposición matricial en la matriz de términos y documentos, la descomposición que se usa es la descomposición de valor singular.
Ahora necesitamos explicar a que se refiere con la descomposición de valor singular, es un método de factorización de matrices que representa una matriz en el producto de dos matrices, para esto usamos una formula muy sencilla:

$$M=U\Sigma V^*$$

- M es una matriz de m × m
- U es una m×n matriz singular izquierda
- Σ es una matriz diagonal n × n con números reales no negativos.
- V es una m×n matriz singular derecha
- V* Es una matriz de n×m, que es la transpuesta de la V.

Wordcloud: Las nubes de palabras, nubes de etiquetas o mosaicos de palabras (en inglés tag cloud o **word cloud**) son un recurso que sirve para presentar una serie de palabras o etiquetas de forma gráfica con distintos colores y tamaños en función de la relevancia de una palabra.

Perplejidad : La perplejidad, utilizada por convención en el modelado del lenguaje, disminuye monótonamente en la probabilidad de los datos de prueba y es algebraicamente equivalente a la inversa de la probabilidad media geométrica por palabra. Una puntuación de perplejidad más baja indica un mejor rendimiento de generalización.

En esencia, dado que la perplejidad es equivalente a la inversa de la media geométrica, una menor perplejidad implica que los datos son más probables. Como tal, a medida que aumenta el número de temas, la perplejidad del modelo debería aumentar.

Marco conceptual:
LSA: Latent Semantic Analysis..
LDA: Latent Dirichlet Allocation.
SVD: Singular Value decomposition.
Word Cloud: Nube de palabras

Desarrollo de proyecto:

Primero empezamos cargando las librerías que usaremos durante el proyecto.

```
import pandas as pd
import numpy as np
import re
import wordcloud
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import CountVectorizer
import matplotlib.pyplot
import warnings
from sklearn.decomposition import LatentDirichletAllocation as LDA
from sklearn.decomposition import TruncatedSVD
from sklearn.decomposition import PCA
```

Cargamos los datos y los imprimimos como un dataframe sin modificaciones.

```
papers = pd.read_csv("papers.csv")
papers.head()
```

	id	year	title	event_type	pdf_name	abstract	paper_text
0	1	1987	Self-Organization of Associative Database and ...	NaN	1-self-organization-of-associative-database-an...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	10	1987	A Mean Field Theory of Layer IV of Visual Cort...	NaN	10-a-mean-field-theory-of-layer-iv-of-visual-c...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	100	1988	Storing Covariance by the Associative Long-Ter...	NaN	100-storing-covariance-by-the-associative-long...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1000	1994	Bayesian Query Construction for Neural Network...	NaN	1000-bayesian-query-construction-for-neural-ne...	Abstract Missing	Bayesian Query Construction for Neural\nNetwor...
4	1001	1994	Neural Network Ensembles, Cross Validation, an...	NaN	1001-neural-network-ensembles-cross-validation...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

vemos que tenemos muchas columnas que no son importantes para el trabajo que estaremos realizando, asi que podemos realizar una limpieza a nuestro dataframe para trabajar solo con los campos necesarios.

```
papers = papers.drop(columns=['id','event_type','pdf_name'])
papers.head()
```

	year	title	abstract	paper_text
0	1987	Self-Organization of Associative Database and ...	Abstract Missing	767\n\nSELF-ORGANIZATION OF ASSOCIATIVE DATABA...
1	1987	A Mean Field Theory of Layer IV of Visual Cort...	Abstract Missing	683\n\nA MEAN FIELD THEORY OF LAYER IV OF VISU...
2	1988	Storing Covariance by the Associative Long-Ter...	Abstract Missing	394\n\nSTORING COVARIANCE BY THE ASSOCIATIVE\n...
3	1994	Bayesian Query Construction for Neural Network...	Abstract Missing	Bayesian Query Construction for Neural\nNetwor...
4	1994	Neural Network Ensembles, Cross Validation, an...	Abstract Missing	Neural Network Ensembles, Cross\nValidation, a...

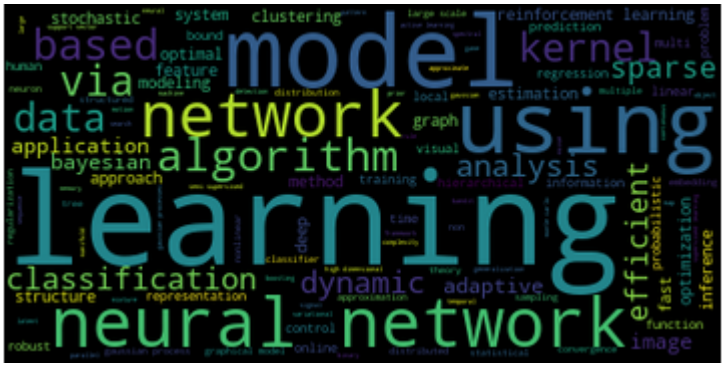
Ya que tenemos los campos específicos que debemos usar vemos que la variable principal title no está normalizada por lo que eliminamos cualquier stopword que tenga y no sean palabras y convertimos todo a minúsculas

```
print(papers['title'].head())
papers['title_processed'] = papers['title'].map(lambda x: re.sub('[,\.!?]', '', x))
papers['title_processed'] = papers['title_processed'].map(lambda x: x.lower())
papers['title_processed'].head()
```

```
0    Self-Organization of Associative Database and ...
1    A Mean Field Theory of Layer IV of Visual Cort...
2    Storing Covariance by the Associative Long-Ter...
3    Bayesian Query Construction for Neural Network...
4    Neural Network Ensembles, Cross Validation, an...
Name: title, dtype: object
0    self-organization of associative database and ...
1    a mean field theory of layer iv of visual cort...
2    storing covariance by the associative long-ter...
3    bayesian query construction for neural network...
4    neural network ensembles cross validation and ...
Name: title_processed, dtype: object
```

Crearemos un wordcloud para ver cuales palabras son las más comunes de manera gráfica de manera visual

```
long_string = ' '.join(papers['title_processed'])
wordcloud = wordcloud.WordCloud()
wordcloud.generate(long_string)
wordcloud.to_image()
```

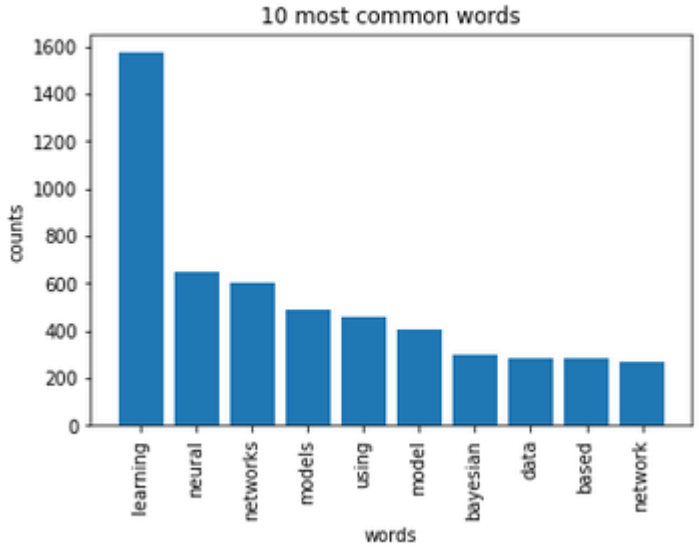


Ahora haremos una barplot de las 10 palabras más comunes de los títulos.

```
def plot_10_most_common_words(count_data, count_vectorizer):
    import matplotlib.pyplot as plt
    words = count_vectorizer.get_feature_names()
    total_counts = np.zeros(len(words))
    for t in count_data:
        total_counts+=t.toarray()[0]

    count_dict = (zip(words, total_counts))
    count_dict = sorted(count_dict, key=lambda x:x[1], reverse=True)[0:10]
    words = [w[0] for w in count_dict]
    counts = [w[1] for w in count_dict]
    x_pos = np.arange(len(words))

    plt.bar(x_pos, counts,align='center')
    plt.xticks(x_pos, words, rotation=90)
    plt.xlabel('words')
    plt.ylabel('counts')
    plt.title('10 most common words')
    plt.show()
count_vectorizer = CountVectorizer(stop_words='english')
count_data = count_vectorizer.fit_transform(papers['title_processed'])
plot_10_most_common_words(count_data, count_vectorizer)
```



Resultados:

Ahora crearemos los dos modelos de prediccion de LDA y LSA, veremos las salidas de los tópicos que nos realiza cada uno de los algoritmos.

```
titulosIda = []
titulosIsa = []
warnings.simplefilter("ignore", DeprecationWarning)
def print_topicsIda(model, count_vectorizer, n_top_words):
    words = count_vectorizer.get_feature_names()
    for topic_idx, topic in enumerate(model.components_):
        print("\nTopic #%d:" %topic_idx)
        print(" ".join([words[i]
                    for i in topic.argsort()[:-n_top_words - 1:-1]]))
        titulosIda.append(" ".join([words[i]
                    for i in topic.argsort()[:-n_top_words - 1:-1]]))

def print_topicsIsa(model, count_vectorizer, n_top_words):
    words = count_vectorizer.get_feature_names()
    for topic_idx, topic in enumerate(model.components_):
        print("\nTopic #%d:" %topic_idx)
        print(" ".join([words[i]
                    for i in topic.argsort()[:-n_top_words - 1:-1]]))
        titulosIsa.append(" ".join([words[i]
                    for i in topic.argsort()[:-n_top_words - 1:-1]]))

number_topics = 5
number_words = 5

lda = LDA(n_components=number_topics)
lda.fit(count_data)

lsa = TruncatedSVD(n_components=number_topics)
lsa.fit(count_data)

print("Topicos encontrados via LDA:")
print_topicsIda(lda, count_vectorizer, number_words)

print("\n")

print("Topicos encontrados via LSA:")
print_topicsIsa(lsa, count_vectorizer, number_words)
```

Métricas:
Aplicamos la métrica de la perplejidad a nuestros tópicos obtenidos la cual viene dentro de la función fit.transform para los tópicos de lda y lsa.

LDA:

```
lda_top=lda.fit_transform(count_data)
print('Perplejidad por cada topico \n')
for i,topic in enumerate(lda_top[0]):
  print("Topico ",i,": ",topic*100)
```

```
Perplejidad por cada topico

Topico  0 :  37.50286511706378
Topico  1 :  3.382209658374495
Topico  2 :  52.38932327293105
Topico  3 :  3.3862706521454218
Topico  4 :  3.339331299485251
```

LSA:

```
lsa_top=lsa.fit_transform(count_data)
l=lsa_top[0]
print('Perplejidad por cada topico \n')
for i,topic in enumerate(l):
  print("Topico ",i," : ",topic*100)
```

```
Perplejidad por cada topico

Topico  0  :  2.5068385834596607
Topico  1  :  3.0021382666304173
Topico  2  :  1.8328660251090008
Topico  3  :  2.536106932747562
Topico  4  :  1.100255908367209
```

Conclusiones:

Con una rapida inspeccion de los resultados se determina que el mejor algoritmo para este caso en el de LSA, ya que en en el cálculo de perplejidad salió con una mejor puntuación en general en comparación de los tópicos de LDA, se obtienen mejores resultados en LSA lo que nos indica que es más probable que aparezcan estos tópicos en los documentos haciéndolos más acertados.

En cuanto a la comparación de wordclouds esta se rige más por razonamiento humano tratando de encontrar una mejor coherencia entre las palabras que aparecen, determinando su importancia por tamaño, en cuanto más grande más importante es la palabra.

En esta comparación se determinó que el wordcloud del algoritmo LSA es más coherente y más acertado en relación a los temas presentes en los documentos.

Por lo tanto en esta comparativa se concluye que es mejor utilizar LSA ya que este algoritmo tiene un mejor desempeño para la identificación de tópicos, ya que este algoritmo tiene un mejor desempeño cuando se trabajan con archivos relativamente pequeños.

Referencias bibliográficas:

- Latent Dirichlet Allocation. (2020, 18 de marzo). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 03:27, marzo 28, 2022 desde https://es.wikipedia.org/w/index.php?title=Latent_Dirichlet_Allocation&oldid=124355588.
- Stochastic Variational Inference", Matthew D. Hoffman, David M. Blei, Chong Wang, John Paisley, 2013
- Cvitanic, T., Lee, B., Song, H. I., Fu, K., & Rosen, D.. *LDA v. LSA: A Comparison of Two Computational Text Analysis Tools for the Functional Categorization of Patents*. *International Conference on Case-Based Reasoning*, (). Retrieved from https://par.nsf.gov/biblio/10055536.
-