Análisis Exploratorio de Datos 2023

U7 - 7A

Análisis exploratorio de documentos de texto mediante LSA

Nombre: David Aaron Ramirez Olmeda
Programa: Maestría en Ciencia de Datos e Información

Introducción:

En el contexto de un análisis de datos nos encontramos con la presencia de datos faltantes en el conjunto de datos de "palmerpenguins". La gestión adecuada de estos valores ausentes es fundamental para garantizar la integridad y la precisión de nuestros resultados. Se presenta un enfoque para abordar y corregir estos datos faltantes, junto con el uso de visualizaciones para mejorar la comprensión de los datos.

Introducción:

En este proyecto, se llevó a cabo un análisis exploratorio de datos y reducción de dimensiones en un conjunto de tweets clasificados según su clase de humor. El objetivo fue identificar patrones temáticos dentro de los tweets utilizando técnicas de procesamiento de lenguaje natural y reducción de dimensiones. A grandes rasgos:

1. Preprocesamiento de Datos:

- Se realizó la carga de datos desde un archivo CSV.
- Se eliminaron las filas con datos faltantes y se aplicó una limpieza básica de texto, incluida la eliminación de stopwords.

2. Vectorización y Reducción de Dimensiones:

- Se utilizó la técnica TF-IDF para convertir los tweets en vectores ponderados.
- Se aplicaron dos métodos diferentes de reducción de dimensiones: Latent Dirichlet Allocation (LDiA) y Latent Semantic Analysis (LSA).

3. Identificación de Temas:

- Con LDiA, se asignaron tópicos dominantes a cada tweet y se realizó un análisis de las palabras más frecuentes en cada tópico.
- Con LSA, se asignaron tópicos dominantes y se visualizó la distribución de estos tópicos por clase.

4. Comparación entre LDiA y LSA:

- Se compararon las distribuciones de tópicos dominantes por clase entre LDiA y LSA.

- Se observó que LSA tendía a asignar más tweets al tópico 0, mientras que LDiA mostraba una distribución más uniforme.

In [22]:
```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer, CountVecto
from sklearn.decomposition import TruncatedSVD, LatentDirichletAllocati
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import Normalizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt
from wordcloud import WordClou
```

1 Genera la matriz de termino documento por clase y por humor_mechanish (tfidf y bag of words)

Notar aquí que no todas las combinaciones se van a poder generar, por ejemplo los no-joke combinados con alguna humor_mechanisms

In [24]:
```
# Carga de datos
df = pd.read_csv('/Users/aaron/Documentos/MCDI/Semestre 2/Análisis Expl

# Eliminar filas con datos faltantes
df = df.dropna(subset=['text'])

# Tokenización, limpieza de texto y eliminación de stopwords
stop_words = set(stopwords.words('spanish'))

def preprocess_text(text):
    words = word_tokenize(text)
    processed_text = ' '.join([word.lower() for word in words if word.i
    return processed_text

df['processed_text'] = df['text'].apply(preprocess_text)

# Análisis de temas por clase y mecanismo de humor
classes = df['klass'].unique()
humor_mechanisms = df['humor_mechanism'].unique()

# Variables para almacenar las matrices no vacías
non_empty_tfidf_matrices = {}
non_empty_bow_matrices = {}

# Función para aplicar TF-IDF
def apply_tfidf(data):
    if data.empty:
        # Retorna una matriz vacía si el conjunto de datos está vacío
        return None

    vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5)

    try:
        tfidf_matrix = vectorizer.fit_transform(data)
        return tfidf_matrix
    except ValueError as e:
        # Captura el error si el vocabulario está vacío
        print(f"Error en TF-IDF: {e}")
        return None

# Función para aplicar Bag of Words
def apply_bow(data):
    if data.empty:
        # Retorna una matriz vacía si el conjunto de datos está vacío
        return None

    vectorizer = CountVectorizer()

    try:
        bow_matrix = vectorizer.fit_transform(data)
        return bow_matrix
    except ValueError as e:
        # Captura el error si el vocabulario está vacío
        print(f"Error en Bag of Words: {e}")
        return None

# Iterar sobre clases y mecanismos de humor
```

```python
for class_label in classes:
    for mechanism_label in humor_mechanisms:
        # Filtrar datos por clase y mecanismo de humor
        subset_data = df[(df['klass'] == class_label) & (df['humor_mech

        # Aplicar TF-IDF
        tfidf_matrix = apply_tfidf(subset_data)
        if tfidf_matrix is not None:
            print(f"TF-IDF Matrix para clase {class_label} y mecanismo
            non_empty_tfidf_matrices[(class_label, mechanism_label)] =

        # Aplicar Bag of Words
        bow_matrix = apply_bow(subset_data)
        if bow_matrix is not None:
            print(f"Bag of Words Matrix para clase {class_label} y meca
            non_empty_bow_matrices[(class_label, mechanism_label)] = bo
```

```
TF-IDF Matrix para clase joke y mecanismo reference:
  (0, 1641)     0.37321034260285707
  (0, 877)      0.37321034260285707
  (0, 75)       0.37321034260285707
  (0, 886)      0.37321034260285707
  (0, 1318)     0.31172205230685923
  (0, 1052)     0.2890285538548455
  (0, 648)      0.3505168441508434
  (0, 799)      0.37321034260285707
  (1, 2016)     0.34339448262512806
  (1, 1382)     0.3225139729759731
  (1, 1501)     0.27200354859632125
  (1, 601)      0.3225139729759731
  (1, 490)      0.22230412795794285
  (1, 2176)     0.34339448262512806
  (1, 2283)     0.5814174000765767
  (1, 85)       0.30769901561072466
  (2, 2362)     0.2183305723071799
  (2, 287)      0.3409305648776627
  (2, 2095)     0.3409305648776627
```

2 Realiza un análisis sobre las frecuencias (por clase y humor_mechanish) de términos sobre los vectores TF-IDF y BoW. Utiliza medidas estadísticas y nubes de palabras

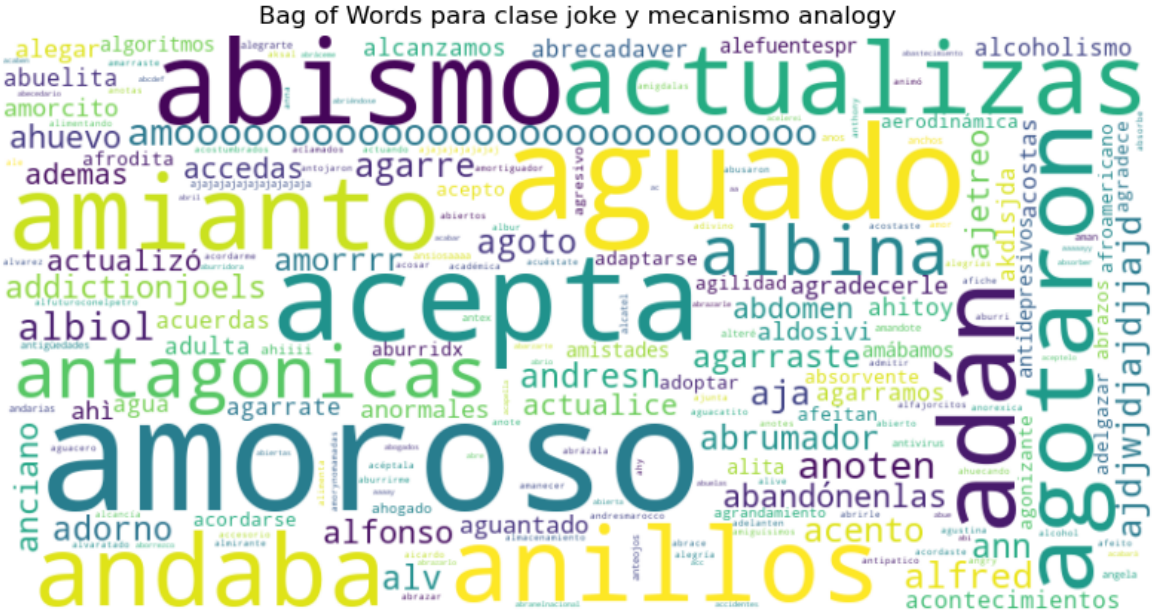
In [29]:
```
# Selecciona una combinación específica de clase y mecanismo
selected_class = 'joke'
selected_mechanism = 'analogy'

# Obtiene la matriz TF-IDF y Bag of Words correspondientes
tfidf_matrix = non_empty_tfidf_matrices.get((selected_class, selected_m
bow_matrix = non_empty_bow_matrices.get((selected_class, selected_mecha

# Obtener el vocabulario original del vectorizador
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5)
vectorizer.fit_transform(df['processed_text'])
feature_names = vectorizer.get_feature_names_out()

# Función para visualizar las frecuencias y generar nubes de palabras
def visualize_frequencies_and_wordcloud(matrix, feature_names, title):
    if matrix is not None:
        # Obtener frecuencias
        frequencies = matrix.sum(axis=0)

        # Visualizar frecuencias
        print(f"Frecuencias para clase {selected_class} y mecanismo {se
        print(f"Media: {frequencies.mean()}, Mediana: {np.median(freque

        # Crear un diccionario de palabras y sus frecuencias
        word_frequencies = {feature_names[i]: frequencies[0, i] for i i

        # Generar y mostrar nube de palabras
        generate_wordcloud(word_frequencies, title)

    else:
        print(f"No hay matriz disponible para clase {selected_class} y

# Función para generar nubes de palabras
def generate_wordcloud(frequencies, title):
    wordcloud = WordCloud(width=800, height=400, random_state=21, max_f
    plt.figure(figsize=(10, 7))
    plt.imshow(wordcloud, interpolation="bilinear")
    plt.title(title)
    plt.axis('off')
    plt.show()

# Visualizar para la matriz TF-IDF
visualize_frequencies_and_wordcloud(tfidf_matrix, feature_names, f"TF-I

# Visualizar para la matriz Bag of Words
visualize_frequencies_and_wordcloud(bow_matrix, feature_names, f"Bag of
```

Frecuencias para clase joke y mecanismo analogy:
[[0.66191005 0.36903489 0.42812296 ... 0.69032061 0.3466308 0.798420
62]]

Media: 0.5199286354212922, Mediana: [[0.28063011 0.29280335 0.3051654
9 ... 0.69032061 0.81937748 6.81808535]], Desviación Estándar: 0.4401
2238007743915

Frecuencias para clase joke y mecanismo analogy:
[[2 1 1 ... 2 1 2]]

Media: 1.5698090692124105, Mediana: [[1. 1. 1. ... 2. 2. 2.]], Desvia
ción Estándar: 1.846565595409933



3. Realiza reducción de dimensión (utilice LatentDirichletAllocation y TruncateSVD)

In [35]:
```
df = pd.read_csv('/Users/aaron/Documentos/MCDI/Semestre 2/Análisis Expl
df = df.dropna(subset=['text'])
df['processed_text'] = df['text'].apply(preprocess_text)

# Crear un vectorizador TF-IDF
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5)
tfidf_matrix = vectorizer.fit_transform(df['processed_text'])

# Reducción de dimensiones con LatentDirichletAllocation
lda_model = LatentDirichletAllocation(n_components=5, random_state=42)
lda_matrix = lda_model.fit_transform(tfidf_matrix)

# Reducción de dimensiones con TruncatedSVD
svd_model = TruncatedSVD(n_components=5, random_state=42)
svd_matrix = svd_model.fit_transform(tfidf_matrix)

# tópicos de LatentDirichletAllocation
topics = lda_model.components_

# componentes de TruncatedSVD
components = svd_model.components_
```

In [36]:
```
# Imprimir algunas estadísticas básicas
print("Dimensiones originales:", tfidf_matrix.shape)
print("Dimensiones después de LDA:", lda_matrix.shape)
print("Dimensiones después de TruncatedSVD:", svd_matrix.shape)

# Imprimir tópicos de LatentDirichletAllocation
print("Tópicos de LatentDirichletAllocation:")
for i, topic in enumerate(topics):
    print(f"Tópico {i+1}: {', '.join([vectorizer.get_feature_names()[ic

# Imprimir componentes de TruncatedSVD
print("\nComponentes de TruncatedSVD:")
for i, component in enumerate(components):
    print(f"Componente {i+1}: {', '.join([vectorizer.get_feature_names(
```

```
Dimensiones originales: (24000, 30494)
Dimensiones después de LDA: (24000, 5)
Dimensiones después de TruncatedSVD: (24000, 5)
Tópicos de LatentDirichletAllocation:
Tópico 1: si, nadie, rt, voy, amor
Tópico 2: si, hoy, nunca, vida, tan
Tópico 3: si, chistes, casa, ser, tan
Tópico 4: si, quiero, ser, vida, voy

/Users/aaron/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/
deprecation.py:87: FutureWarning: Function get_feature_names is depre
cated; get_feature_names is deprecated in 1.0 and will be removed in
1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)

Tópico 5: si, sueño, quiero, días, cosas

Componentes de TruncatedSVD:
Componente 1: si, quiero, vida, voy, ser
Componente 2: quiero, ser, día, vida, ir
Componente 3: quiero, si, ir, quieres, helado
Componente 4: día, hoy, dormir, buen, si
Componente 5: vida, amor, siempre, alguien, toda
```

4. Elije un número de tópicos de al menos el mismo numero de humor_mechanish+1

Tenemos 13, por lo que podemos usar 15

In [39]:
```
# Obtener el número único de humor_mechanisms
num_humor_mechanisms = len(df['humor_mechanism'].unique())
num_humor_mechanisms
```

Out[39]: 13

5. Determina el tópico de cada tweet y basado en los grupos obtenidos para cada tema, identifica las 10 palabras más frecuentes en cada tema.

In [48]:
```
df = pd.read_csv('/Users/aaron/Documentos/MCDI/Semestre 2/Análisis Expl
df = df.dropna(subset=['text'])
df['processed_text'] = df['text'].apply(preprocess_text)

# Crear un vectorizador TF-IDF
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5)
tfidf_matrix = vectorizer.fit_transform(df['processed_text'])

# Reducción de dimensiones con LatentDirichletAllocation
num_topics = 15
lda_model = LatentDirichletAllocation(n_components=num_topics, random_s
lda_matrix = lda_model.fit_transform(tfidf_matrix)

# Asignar el tópico dominante a cada tweet
df['dominant_topic'] = lda_matrix.argmax(axis=1)

# Imprimir las 10 palabras más frecuentes en cada tópico
print("Palabras más frecuentes en cada tópico:")
for i, topic in enumerate(lda_model.components_):
    top_words_indices = topic.argsort()[:-11:-1]
    top_words = [vectorizer.get_feature_names()[idx] for idx in top_wor
    print(f"Tópico {i+1}: {', '.join(top_words)}")
```

```
Palabras más frecuentes en cada tópico:
Tópico 1: si, rt, favor, gracias, ojos, hoy, acá, chistes, hace, vida
Tópico 2: si, hoy, día, vida, nunca, extraño, feliz, siempre, mas, do
mingo

/Users/aaron/opt/anaconda3/lib/python3.9/site-packages/sklearn/utils/
deprecation.py:87: FutureWarning: Function get_feature_names is depre
cated; get_feature_names is deprecated in 1.0 and will be removed in
1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

Tópico 3: si, ser, día, chistes, buen, mañana, gente, vida, hacer, puedo
Tópico 4: si, mejor, vida, ser, voy, buena, quiero, bien, hacer, día
Tópico 5: si, voy, así, dormir, da, día, vida, bueno, duermo, quiero
Tópico 6: si, lunes, amor, mierda, hace, quiero, falta, tan, chistes, mejor
Tópico 7: si, sé, hoy, chistes, bien, ahora, día, ser, tan, dice
Tópico 8: tan, chistes, si, odio, aeroméxicotienepésimoservicio, puta, nunca, rt, problema, madre
Tópico 9: si, amor, vida, chistes, va, mamá, mujer, quiero, hace, voy
Tópico 10: si, semana, vez, bien, fin, ser, quiero, puedo, buenas, llorar
Tópico 11: si, va, chistes, voy, tan, dice, bien, vida, mamá, verano
Tópico 12: amo, si, amor, chistes, vida, tan, años, voy, amigos, ver
Tópico 13: si, quiero, alguien, dormir, ir, ganas, siempre, voy, hola, casa
Tópico 14: días, buenos, mal, si, amor, siento, hoy, ser, dias, hermosa
Tópico 15: si, sueño, dia, bien, quiero, nunca, va, siempre, voy, ser

6. Determina si es posible identificar los temas en cada una de las clases y compara los resultados entre LSA y LDiA.

In [67]:
```
# Reducción de dimensiones con LatentDirichletAllocation
num_topics = 15
lda_model = LatentDirichletAllocation(n_components=num_topics, random_s
ldia_matrix = lda_model.fit_transform(tfidf_matrix)

# Asignar el tópico dominante a cada tweet utilizando LDiA
ldia_dominant_topic = ldia_matrix.argmax(axis=1)
df['ldia_dominant_topic'] = ldia_dominant_topic

# Crear un DataFrame auxiliar con información de clase y tópico dominar
df_aux = pd.DataFrame({
    'klass': df['klass'],
    'ldia_dominant_topic': df['ldia_dominant_topic']
})

# Contar la frecuencia de tópicos dominantes por clase
topic_distribution_by_class_ldia = df_aux.groupby(['klass', 'ldia_domir

# Imprimir la distribución de tópicos dominantes por clase
print("\nDistribución de tópicos dominantes por clase (LDiA):")
print(topic_distribution_by_class_ldia)
```

```
Distribución de tópicos dominantes por clase (LDiA):
ldia_dominant_topic   0    1    2     3     4    5    6    7    8
9    10  \
klass
joke                534  489  581   758   708  564  531  677  587
693  547
no-joke             885  907  860  1112  1082  915  817  954  904  1
227  783

ldia_dominant_topic   11    12   13   14
klass
joke                671   897  457  559
no-joke             975  1518  853  955
```

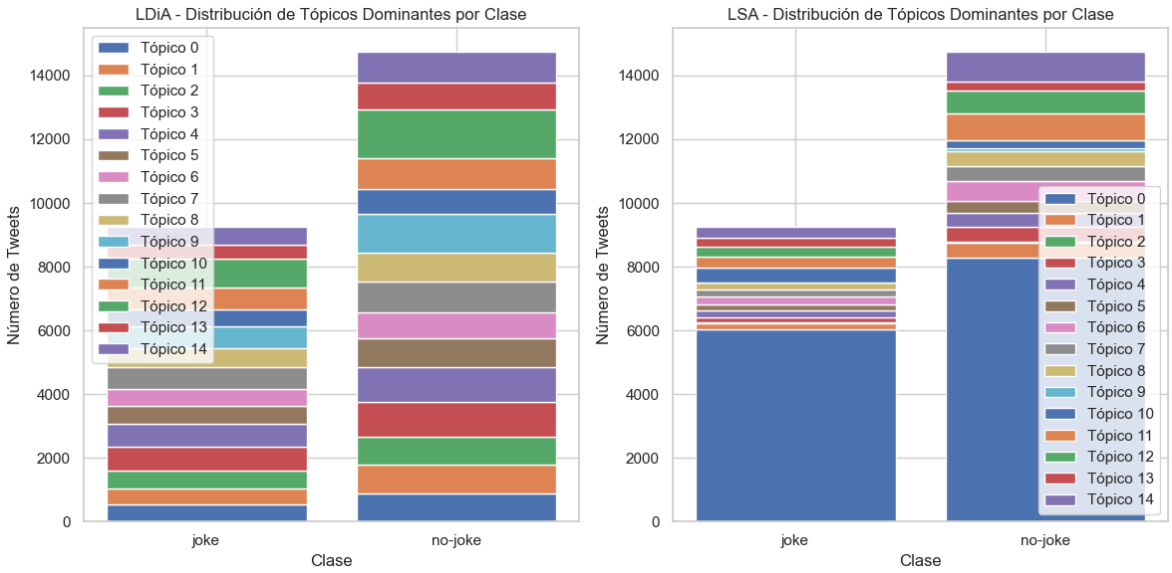
In [68]:

```
df = pd.read_csv('/Users/aaron/Documentos/MCDI/Semestre 2/Análisis Expl
df = df.dropna(subset=['text'])
df['processed_text'] = df['text'].apply(preprocess_text)

# Crear un vectorizador TF-IDF
vectorizer = TfidfVectorizer(sublinear_tf=True, max_df=0.5)
tfidf_matrix = vectorizer.fit_transform(df['processed_text'])

# Reducción de dimensiones con LSA
num_topics = 15
lsa_model = TruncatedSVD(n_components=num_topics, random_state=42)
lsa_matrix = lsa_model.fit_transform(tfidf_matrix)

# Asignar el tópico dominante a cada tweet
df['lsa_dominant_topic'] = lsa_matrix.argmax(axis=1)

# Crear un DataFrame auxiliar con información de clase y tópico dominar
df_aux = pd.DataFrame({'klass': df['klass'], 'lsa_dominant_topic': df['

# Contar la frecuencia de tópicos dominantes por clase
topic_distribution_by_class_lsa = df_aux.groupby(['klass', 'lsa_dominar

# Imprimir la distribución de tópicos dominantes por clase
print("\nDistribución de tópicos dominantes por clase (LSA):")
print(topic_distribution_by_class_lsa)
```

```
Distribución de tópicos dominantes por clase (LSA):
lsa_dominant_topic    0    1   2    3    4    5    6    7    8   9
10   11  \
klass
joke                6039  190  12  167  228  185  255  222  190  27
451  338
no-joke             8295  459  41  460  425  394  601  492  462  76
273  831

lsa_dominant_topic   12   13   14
klass
joke                334  280  335
no-joke             719  298  921
```

In [71]:

```
import matplotlib.pyplot as plt

# Configuración de las clases y tópicos
classes = topic_distribution_by_class_ldia.index
topics_ldia = topic_distribution_by_class_ldia.columns
topics_lsa = topic_distribution_by_class_lsa.columns

# Crear barras apiladas para LDiA
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
bottom_ldia = None
for i, topic in enumerate(topics_ldia):
    plt.bar(classes, topic_distribution_by_class_ldia[topic], label=f'T
    bottom_ldia = (
        topic_distribution_by_class_ldia[topics_ldia[:i+1]].sum(axis=1)
        else bottom_ldia + topic_distribution_by_class_ldia[topic]
    )
plt.title('LDiA - Distribución de Tópicos Dominantes por Clase')
plt.xlabel('Clase')
plt.ylabel('Número de Tweets')
plt.legend()

# Crear barras apiladas para LSA
plt.subplot(1, 2, 2)
bottom_lsa = None
for i, topic in enumerate(topics_lsa):
    plt.bar(classes, topic_distribution_by_class_lsa[topic], label=f'Tó
    bottom_lsa = (
        topic_distribution_by_class_lsa[topics_lsa[:i+1]].sum(axis=1) i
        else bottom_lsa + topic_distribution_by_class_lsa[topic]
    )
plt.title('LSA - Distribución de Tópicos Dominantes por Clase')
plt.xlabel('Clase')
plt.ylabel('Número de Tweets')
plt.legend()

# Ajustar diseño y mostrar gráfico
plt.tight_layout()
plt.show()
```



En la visualización de los resultados, se observa que en el modelo LSA, el tópico 0 tiene una presencia notable y parece dominar una proporción sustancial de los tweets, posiblemente representando más de la mitad del total. Por otro lado, en el modelo LDiA, la distribución de tópicos es más equitativa, indicando una distribución más uniforme de tópicos dominantes en los tweets.

Conclusiones:

- La aplicación de técnicas de reducción de dimensiones permitió identificar patrones temáticos en los tweets de humor.
- LDiA y LSA proporcionaron perspectivas diferentes, destacando la importancia de explorar múltiples enfoques en análisis de texto.
- La distribución de tópicos por clase puede variar según el método de reducción de dimensiones utilizado.

Referencias:

[1] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent Dirichlet Allocation. Journal of Machine Learning Research, 3(Jan), 993–1022.

[2] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., & Harshman, R. (1990). Indexing by Latent Semantic Analysis. Journal of the American Society for Information Science, 41(6), 391–407.

[3] Notas del curso.