



**INSTITUTO POLITÉCNICO NACIONAL**



**ESCUELA SUPERIOR DE CÓMPUTO**

## **AGRUPAMIENTO DE TEXTOS**

**PROFESOR:**

**ENRIQUE ALFONSO CARMONA GARCÍA**

**ALUMNOS:**

**ITURIEL ENRIQUE FLORES ESTRADA**

**7BM1**

**FECHA DE ENTREGA: 21 DE ABRIL 2023**

## ÍNDICE

Introducción .....	3
EXTRACCIÓN DE DATOS.....	3
Limpieza de datos.....	3
K-MEANS .....	5
Vectorización de datos :TF-IDF.....	5
K-MEANS:PRUEBA DE K.....	5
Resultados con valores aleatorios.....	6
cluster optimo con la tecnica de codo .....	7
GRAFICA .....	9
Capa de enbeding.....	10
Resultados.....	11
cONCLUSIONES.....	12

## INTRODUCCIÓN

En esta práctica, se explorará el uso de la técnica de agrupamiento K-means en conjunto con tf-idf para analizar un conjunto de datos de noticias en inglés. El conjunto de datos cuenta con dos columnas: texto y etiqueta, y en total contiene 7000 observaciones, con 2000 observaciones por cada clase. El objetivo de la práctica es utilizar la técnica de agrupamiento K-means para analizar los patrones en los datos y agrupar las noticias en distintas categorías. Para ello, se empleará tf-idf para representar la vectorización de cada palabra, y así poder aplicar la técnica de agrupamiento K-means. De esta manera, se espera obtener una mayor comprensión de los patrones y tendencias en las noticias, lo que permitirá identificar y categorizar información relevante de manera más efectiva.

Así mismo también se realiza la capa de embedding junto la representación de los documentos resultante de la técnica de "Hot-Encoding"

## EXTRACCIÓN DE DATOS

Para poder realizar el modelado de los algoritmos de agrupación, se utilizaron un total de 6000 datos donde cada clase tenía 2000 observaciones en este caso como eran noticias se tomaron en cuenta world, sports, business y esto se concatenaron para tener los datos seleccionados con los que estarían trabajando como se muestra en la *imagen*. Clases y observaciones

```
#Leer archivo
df = pd.read_csv('train.csv')
# Agrupar los datos por categoría y seleccionar los primeros 2000 de cada grupo
df['label'] = df['label'].astype(str)

#2000 observaciones para cada clase
world = df[df['label'] == '1'].head(2000)
Sports = df[df['label'] == '2'].head(2000)
Business = df[df['label'] == '3'].head(2000)

#Se junta las selecciones de cada
df=pd.concat([world, Sports, Business], ignore_index=True)
```

### 1. Clases y observaciones

## LIMPIEZA DE DATOS

En este caso como los datos están en inglés se utilizó la librería de spacy y nltk para poder realizar el stemming y la lematización

En la primera estructura del código se utilizan expresiones regulares para poder limpiar primero los números, signos de puntuación y acentos.

En el caso de la línea donde se utiliza la función *unicodate* lo que hace es para eliminar o transformar caracteres no ASCII en una cadena de texto esto también para reducir el ruido y asegurar la compatibilidad

En la ultima parte que se muestra en la imagen “2.Limpieza de datos”,se convierte el texto en minisculas

```
clean_text = n_text
#Eliminar numeros
clean_text = re.sub('\d', '', clean_text)
# Eliminar signos de puntuación
clean_text = re.sub('[^\w\s]', '', clean_text)

# Acentos
clean_text = unicodedata.normalize('NFKD', clean_text).encode('ascii', 'ignore').decode('utf-8')

# Convertir texto en minisculas
clean_text = clean_text.lower()
```

## 2.Limpieza de datos

En la imagen “3.Normalización de datos”,se utilizan funciones que nos proporcionan las bibliotecas ya mencionadas

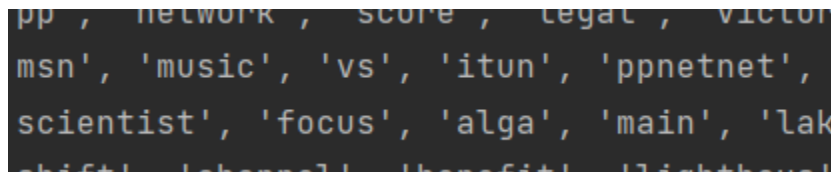
1. `clean_text = nlp(clean_text)`: esta línea utiliza la biblioteca de procesamiento de lenguaje natural (NLP) de Spacy para analizar el texto contenido en la variable `clean_text`.
2. `tokens = [stemmer.stem(token.lemma_) if token.lemma_ != '-PRON-' else stemmer.stem(token.text) for token in clean_text if not token.is_stop and not token.is_punct and not token.is_space]`: esta línea realiza la lematización y stemming del texto analizado en la línea anterior. En esta línea, cada token en el texto analizado se procesa mediante una comprensión de lista (list comprehension) que aplica la lematización y el stemming a cada token. La comprensión de lista filtra también los tokens que son signos de puntuación o espacios en blanco, o que son palabras de parada ("stop words") (determinadas por Spacy y marcadas con la propiedad `is_stop`). Si un token es un pronombre (marcado por Spacy con la etiqueta `-PRON-`), se utiliza su texto original en lugar de su lema.

```
# Analizar texto completo con Spacy
clean_text = nlp(clean_text)
# Lematizar y Stemming
tokens = [stemmer.stem(token.lemma_) if token.lemma_ != '-PRON-' else stemmer.stem(token.text) for token in clean_text if not token.is_stop and not token.is_punct]
print(tokens)
documento = ' '.join(tokens)
```

## 3.Normalización datos

Ahora lo que retornara es una lista de tokens ya normalizados

Como se muestra en la “Imagen 4 Resultados Normalización de datos”



## 4.Resultados Normalización

## K-MEANS

Ya normalizados con la función creada anteriormente llamada `def normaliza(text)`: creada en el archivo `preprocesamiento.py`

Se importa primero la función creada como se muestra en la imagen 5

```
#Aplicamos normalización de texto
df['Documento_Normalizado'] = df['text'].apply(normalizar)
```

5.Aplicación normalización de datos

## VECTORIZACIÓN DE DATOS :TF-IDF

Después ya haber obtenido la normalización de los datos se procede a vectorizar los datos en este caso se esta utilizando TF-IDF en este caso esta utilizando la función de la biblioteca que tiene implementada `sklearn.feature_extraction.text` como se muestra en la imagen “6.Tf-idf”

```
# crear el objeto vectorizador
vectorizer = TfidfVectorizer()

# ajustar y transformar los documentos del DataFrame
X = vectorizer.fit_transform(df['Documento_Normalizado'])
```

6.Tf-idf

## K-MEANS:PRUEBA DE K

Ya vectorizado los datos se crea el modelo de k means para poder testear con diferentes cluster como se puede mostrar en la *imagen “7.Lista al azar de clusters”*.En este caso se esta realizando una lista de longitud n que va a tener valores de entre 3,15,que será nuestros valores en la función de `k-means`

```
def valores_s(n):
    valores_salida = []
    for i in range(n):
        valor = random.randint(2, 8)
        if valor not in valores_salida:
            valores_salida.append(valor)

    valores_salida.sort(reverse=True)

    return valores_salida
```

#### 7.Lista aleatoria de clusters

Primero, se crea una lista aleatoria de valores "k" utilizando una función llamada "valores\_s" como se muestra en la imagen de arriba *"5.Lista aleatoria de clusters"*

Luego, se inicializa una lista vacía llamada "inertia\_scores", que se utilizará para almacenar la inercia de cada modelo de KMeans ajustado como se muestra en la la imagen *"8.Clusters"*

A continuación, el código itera a través de la lista de valores "k" utilizando un bucle "for". En cada iteración, se inicializa un objeto de modelo KMeans con el número de clústeres "k" actual y se ajusta el modelo a los datos "X". Después, se calcula la inercia del modelo utilizando el método "inertia\_" y se agrega a la lista "inertia\_scores".

```
#Creación de lista al azar
k_list=valores_s(n)
# Inicializar listas vacías para almacenar los resultados
inertia_scores = []

# Iterar sobre el número de clusters
for k in k_list:
    # Inicializar modelo KMeans
    kmeans = KMeans(n_clusters=k, random_state=42)

    # Ajustar el modelo a los datos
    kmeans.fit(X)

    # Calcular inercia
    inertia_scores.append(kmeans.inertia_)
print(k_list,inertia_scores)
```

#### 8.Clusters

### RESULTADOS CON VALORES ALEATORIOS

Como se muestra en la imagen *"9.Resultados con valores aleatorios"*,Se puede notar que se tienen los siguientes clusters que se probaron :6,8,13,3,10

Esta lista muestra el puntaje de silueta para cada cluster en el conjunto de datos. El puntaje de silueta mide cuán similar es un punto de datos a su propio cluster en comparación con otros clusters. Cuanto mayor sea el

puntaje de silueta, mejor será la calidad del clustering. Por ejemplo, si la lista de puntajes de silueta es [5792.910420206184, 5764.6000111057465, 5704.332662415239, 5866.4037346718305, 5737.171877825104], significa que hay 5 clusters y que el tercer cluster tiene el puntaje de silueta más alto (5704.33), lo que indica que es un cluster bien definido y los puntos de datos en él están muy separados de los puntos de datos en otros clusters. Por otro lado, el primer y segundo cluster tienen puntajes de silueta similares (5792.91 y 5764.60), lo que indica que puede haber una superposición o ambigüedad entre ellos.

```
[6, 8, 13, 3, 10] [5792.910420206184, 5764.6000111057465, 5704.332662415239, 5866.4037346718305, 5737.171877825104]
```

## 9.Resultados con valores aleatorios

### CLUSTER OPTIMO CON LA TECNICA DE CODO

En la siguiente imagen “Codo  $k_{optimo}$ ” que se muestra busca el  $k$  optimo para la implementación :

La variable `deltas` está calculando la diferencia entre los valores de inercia consecutivos, donde la inercia es una medida de cómo los puntos de datos están dispersos alrededor del centroide en un cluster.

La variable `slopes` está calculando la pendiente entre cada par de puntos de datos consecutivos. Estas pendientes representan la tasa de cambio en la inercia a medida que aumenta el número de clusters.

La variable `k_optimo` está seleccionando el número de clusters que corresponde a la pendiente más grande. Este valor se obtiene a través del índice de la lista de pendientes (`slopes`) donde se encuentra el valor máximo de la lista.

```
deltas = np.diff(inertia_scores)
slopes = [deltas[i] / deltas[i - 1] for i in range(1, len(deltas))]
k_optimo = k_list[slopes.index(max(slopes))]
```

## 10.Codo $K_{optimo}$

Como se puede mostrar en la imagen “11.Ajuste de *K-means*”, lo que se hace en la primera realiza un análisis de clustering en un conjunto de datos dado “data” utilizando el algoritmo de *K-means*.

La función primero llama a otra función llamada “*k-meas*” con la entrada “data” y el valor “5”. Esta función probablemente determina el número óptimo de clusters “*k*” para el conjunto de datos “data” utilizando algún tipo de métrica o heurística.

Luego, la función crea un objeto *KMeans* con el número óptimo de clusters “*kopt*” y lo ajusta a los datos utilizando el método `fit()`.

Se asigna etiquetas a cada punto de datos utilizando el método `predict()` del objeto *KMeans*, luego función cuenta el número de puntos de datos en cada cluster utilizando las funciones `unique()` y `count()` del módulo `numpy` y muestra el resultado como un diccionario.

La función calcula el puntaje de silueta para el clustering utilizando la función `silhouette_score()` del módulo `sklearn` y muestra el resultado

```
def ajuste(data):
    kopt=k_means(data,5)
    kmeans = KMeans(n_clusters=kopt, random_state=42)
    kmeans.fit(data)

    # Asignar etiquetas a cada muestra
    labels = kmeans.predict(data)

    # Imprimir el número de muestras en cada cluster
    unique, counts = np.unique(labels, return_counts=True)
    print(dict(zip(unique, counts)))

    # Calcular el score de silueta
    silhouette_avg = silhouette_score(data, labels)
    print("El score de silueta es:", silhouette_avg)
```

### 11.Ajuste de k-means

En la imagen “12.Grafica Cluster” lo que se hace es graficar los clusters

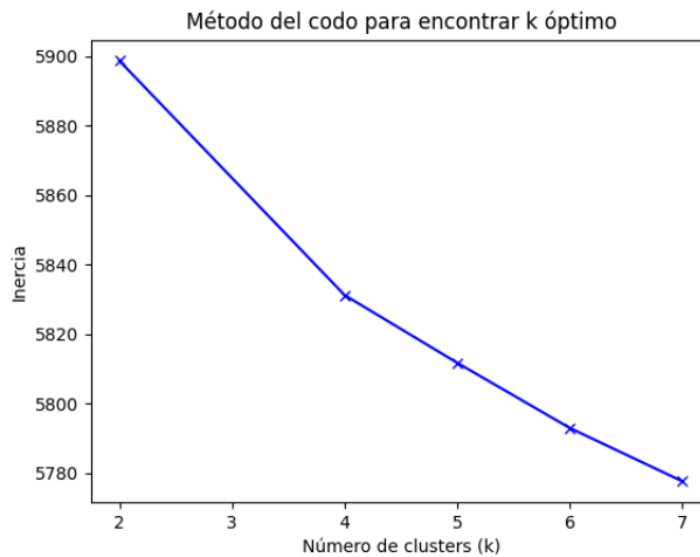
```
# Reducir dimensionalidad a 2
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X.toarray())

# Graficar clusters
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels)
plt.title('Clusters con K Optimo = %d' % kopt)
plt.show()
```

### 12.Graficar Cluster

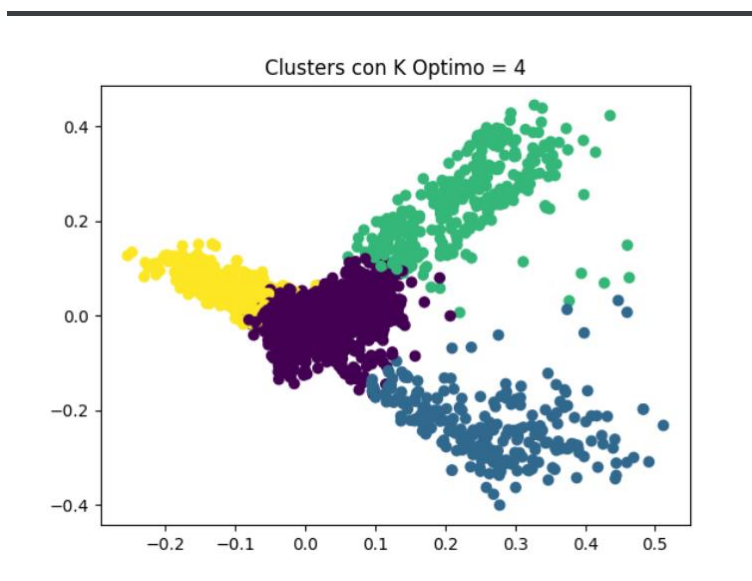


## GRAFICA



### 13.Codo grafica

En este caso en la grafica se puede observar que el optimo es 4 asi que se puede notar en imagen 13 *Grafica de clusters*



### 14.Grafica Clusters

Podemos observar en la imagen 14, que la representación gráfica de los clusters muestra una distribución clara y bien definida de los grupos de datos. Aunque es cierto que en algunos puntos los clusters verde y morado se superponen, se puede notar claramente la diferencia entre cada uno de ellos. Esto indica que el algoritmo de agrupamiento utilizado fue efectivo en la separación de los datos en distintos grupos, lo que

permitirá una mayor comprensión y análisis de los patrones y tendencias presentes en los datos. Es importante destacar que la visualización de los clusters es una herramienta útil para identificar patrones y tendencias en los datos, lo que permite la toma de decisiones informadas y eficaces.

## CAPA DE EMBEDDING

En este caso con la creación de la capa de embedding se tuvieron problemas debido a que la misma función de los modelos no permite que no sean sin etiquetas y al ser un modelo no supervisado

En este caso lo primero que se hizo fue leer el archivo y después se aplica la normalización del texto después se utilizó `onehot_encoder` para realizar hot-encoding,

```
# Leer noticias
df = pd.read_csv('train.csv')
# Normalización del texto
df['Documento_Normalizado'] = df['Contenido'].apply(normalizar)

# Crear un objeto CountVectorizer
vectorizer = CountVectorizer(binary=True, max_features=1000)

# Ajustar el vectorizador a tus datos
vectorizer.fit(df['Documento_Normalizado'])

# Obtener la matriz de representaciones de palabras
X = vectorizer.transform(df['Documento_Normalizado'])

# Crear el objeto OneHotEncoder
onehot_encoder = OneHotEncoder(sparse=True)
```

### 15.hot-encoding

En este caso como se muestra en la capa de embedding “15.Capa embadig” después de haber aplicado el hot-encoding, se convierte la matriz dispersa en una matriz densa y redimensiona la matriz para que tenga la forma (número de documentos, longitud máxima de las secuencias, número de palabras únicas en el vocabulario).

Crea una capa de embedding con la dimensión del espacio de embedding igual a 10.

Genera una lista de 5 valores enteros aleatorios entre 3 y 10, que se utilizarán como el número de unidades de la capa de salida de la red neuronal.

Para cada valor de la lista generada en el paso anterior, crea un modelo de redes neuronales que consiste en la capa de embedding, una capa Flatten que aplanará la matriz de salida del embedding, una capa Dense con 16 unidades y una función de activación relu, y una capa Dense con el número de unidades generado aleatoriamente y una función de activación softmax.,compila el modelo con el optimizador Adam y la función de pérdida dummy\_loss.

Entrena el modelo durante 10 épocas utilizando la matriz de entrada `X_onehot_sparse` y una matriz y aleatoria.

Obtiene la matriz de embedding de la capa de embedding y utiliza la técnica de reducción de dimensionalidad PCA para visualizarla en un gráfico scatter.

```
def modelo(data):
    # Dimensión del espacio de embedding
    embedding_dim = 10

    # Longitud máxima de las secuencias
    max_length = 50

    # Convertir la matriz dispersa en una matriz densa y redimensionarla
    data = data.toarray().reshape(data.shape[0], max_length, -1)

    # Crear la capa de embedding
    embedding_layer = Dense(units=embedding_dim, activation='linear', input_shape=(max_length, data.shape[2]))
    salida_embedding = valores_s(5)
    for valor in salida_embedding:
        # Crear un modelo de redes neuronales
        model = Sequential([
            embedding_layer, # Capa de embedding
            Flatten(), # Aplanar la matriz de salida del embedding
            Dense(16, activation='relu'),
            Dense(valor, activation='softmax') # Capa de salida con 4 unidades y función de activación softmax
        ])

        # Compilar el modelo
        model.compile(optimizer='adam', loss=dummy_loss)

        # Entrenar el modelo
        y = np.random.uniform(size=(data.shape[0], 1))
        model.fit(data, y, epochs=10)
```

### 16.Capa Embedding

.Como se puede notar se quieren hacer etiquetas aleatorias esto debido a que para la creación de esta capa se requiere que se tengan las etiquetas pero para ello esto no sería como tal un modelo no supervisado además que los resultados no serían óptimos .Aunque se trata de hacer etiquetas se podría decir falsas aun así el mismo modelo requiere que exista una función de pérdida en este caso se creó la de dummy\_loss que no es necesario tener las etiquetas

Ya por último lo que se busca es que los vectores de salida que se tiene de la capa de embedding se agrupen. En este caso se quiso intentar por medio de PCA para la visualización de los mismos

```
model.fit(data, y, epochs=10)

# Obtener la matriz de embedding
embedding_matrix = embedding_layer.get_weights()[0]

pca = PCA(n_components=2)
embedding_matrix_pca = pca.fit_transform(embedding_matrix)

plt.scatter(embedding_matrix_pca[:, 0], embedding_matrix_pca[:, 1])
plt.show()

print('Matriz de embedding:', embedding_matrix)

return embedding_matrix
```

### 17.Resultados Capa embedding

## RESULTADOS

En este caso no se pudo llegar al objetivo que se quería llegar porque como se menciona es necesario tener las etiquetas para poder generar la capa de embedding y como se esta hablando de que es un modelo no supervisado en teoría no se deberían requerir de ellas.

Se puede hacer este tipo de agrupación pero también implementando lo que es k-means .

## CONCLUSIONES

En resumen, en la práctica se logró ejecutar el algoritmo de k-means con un resultado óptimo utilizando un valor de k de 4, a pesar de que el método del código sugiriera un valor de 8. Aunque las agrupaciones no se veían completamente separadas, la visualización mostraba una buena distribución de los datos. Sin embargo, en la segunda parte de la práctica, no se pudo llegar al resultado esperado debido a la falta de etiquetas para la creación de la capa de embedding, lo que generó errores en la ejecución del modelo no supervisado. En conclusión, aunque se logró obtener buenos resultados con el algoritmo de k-means, se evidenció la importancia de contar con etiquetas para la creación de modelos supervisados.

Ante la falta de etiquetas para la creación de la capa de embedding, se intentó crear etiquetas falsas o aleatorias. Aunque esto puede parecer una solución viable, es importante destacar que la agrupación resultante puede no ser óptima para la tarea específica que se está tratando de realizar. La agrupación que se obtiene de la capa de embedding es real en el sentido de que es una representación vectorial de los datos de entrada que captura algunas de sus relaciones y similitudes. Sin embargo, esta agrupación puede no tener una interpretación directa y puede no ser óptima para la tarea específica que se está tratando de realizar. En otras palabras, aunque las etiquetas se hayan utilizado para entrenar la capa de embedding, la agrupación resultante puede seguir siendo considerada no supervisada si no se utiliza en un contexto de clasificación o predicción.

A pesar de los esfuerzos por crear etiquetas falsas o al azar para la capa de embedding, no se pudo llegar al resultado esperado debido a que esta capa requiere etiquetas precisas para su entrenamiento. En este caso, al trabajar con modelos no supervisados, no se tenían las etiquetas necesarias para completar esta parte de la práctica, lo que generó errores en la ejecución del modelo y no permitió obtener una agrupación óptima.