Programa académico: Ingeniería en Inteligencia artificial
Unidad de aprendizaje : Aplicaciones de NLP
Nombre del Alumno:_____
Fecha de entrega:_____
Calificación:_____

Practica de laboratorio 1: Aplicaciones Agrupación de textos y vectorización de palabras.

A) Marco teórico

El algoritmo MultinomialNB se utiliza comúnmente en tareas de clasificación de texto, especialmente cuando se trata de contar frecuencias de palabras o términos en documentos.

- **Teorema de Bayes**: El teorema de Bayes es un principio fundamental en estadísticas y probabilidad que se utiliza para calcular la probabilidad condicional. En el contexto de la clasificación de texto, se utiliza para estimar la probabilidad de que un documento pertenezca a una clase particular dada su contenido.
- **Modelo Multinomial Naive Bayes**: El Modelo MultinomialNB asume que las características (palabras en este caso) son generadas por una distribución multinomial. Es *ingenuo* en el sentido de que asume que las características (palabras) son independientes entre sí, lo que puede no ser cierto en la práctica. A pesar de esta suposición simplificada, MultinomialNB funciona sorprendentemente bien en tareas de clasificación de texto.

A continuación, se presenta una descripción matemática simplificada del algoritmo Multinomial Naive Bayes:

Dado un conjunto de características (palabras) X = {x_1, x_2, ..., x_n}, y una clase C_k, donde k representa la clase (por ejemplo, positiva o negativa):

1. **Probabilidad a priori (Prior Probability - P(C_k))**: Se calcula la probabilidad a priori de la clase C_k, que representa la probabilidad de que un documento pertenezca a la clase k. Esto se calcula como la proporción de documentos en la clase k en el conjunto de entrenamiento.
   P(C_k) = (Número de documentos en la clase k) / (Número total de documentos)
2. **Probabilidad de verosimilitud (Likelihood - P(X|C_k))**: Se calcula la probabilidad de verosimilitud de las características dadas la clase C_k. Esto se hace asumiendo una distribución multinomial de las características (palabras).
   P(X|C_k) = Π [P(x_i|C_k)] para i = 1 a n
   Donde P(x_i|C_k) se calcula como la frecuencia de la palabra x_i en documentos de clase

C_k dividida por la suma de las frecuencias de todas las palabras en documentos de clase C_k.

3. **Probabilidad posterior (Posterior Probability - P(C_k|X))**: Se utiliza el teorema de Bayes para calcular la probabilidad posterior de que un documento pertenezca a la clase C_k dado el conjunto de características X.
   P(C_k|X) = [P(C_k) * P(X|C_k)] / P(X)
   P(X) se puede calcular como la suma de [P(C_k) * P(X|C_k)] para todas las clases.
4. **Clasificación**: Se asigna la clase con la mayor probabilidad posterior como la clase predicha para el documento.

Con respecto a metodologías de agrupamiento, K-vecinos cercanos del inglés K-Nearest Neighbors (KNN), este un algoritmo de clasificación y agrupación que se puede utilizar en el procesamiento de lenguaje natural (NLP) para tareas como la agrupación de reseñas de texto en categorías como positivas y negativas. Aunque KNN es más comúnmente utilizado como un algoritmo de clasificación, también puede aplicarse en tareas de agrupación cuando se utiliza con técnicas de reducción de dimensionalidad, como TF-IDF o Word2Vec.

La idea central detrás del algoritmo KNN es bastante simple:

1. **Definir una medida de similitud**: En el contexto de NLP, esta medida de similitud podría ser la distancia coseno entre los vectores de características de dos documentos. Los documentos más similares tendrán una distancia coseno más cercana a 1, mientras que los documentos menos similares tendrán una distancia coseno más cercana a 0.
2. **Seleccionar un valor para K**: K representa el número de vecinos más cercanos que se utilizarán para tomar una decisión. Por ejemplo, si K = 5, significa que se tomarán en cuenta las etiquetas de las 5 reseñas más cercanas para decidir la etiqueta de una reseña nueva.
3. **Para clasificación**: Cuando se utiliza KNN para clasificación de texto, se toma una nueva reseña y se calcula su similitud con todas las reseñas en el conjunto de entrenamiento. Luego, se seleccionan las K reseñas más cercanas en función de la medida de similitud. La etiqueta más común entre esos K vecinos se asigna como la etiqueta de la nueva reseña.
4. **Para agrupación**: Si deseas utilizar KNN para agrupar reseñas de texto en grupos similares, puedes aplicar técnicas de reducción de dimensionalidad (como PCA) para reducir la dimensionalidad de los vectores de características y luego aplicar KNN para agrupar documentos similares en grupos basados en la distancia entre los vectores reducidos.

KNN es un algoritmo flexible y fácil de entender, pero tiene algunas limitaciones, como la sensibilidad a valores atípicos y el aumento en la complejidad computacional con un gran conjunto de datos. Además, la elección de K y la medida de similitud son importantes y deben ajustarse de manera adecuada para obtener buenos resultados en una tarea específica de NLP.

Como se mencionó Word2Vec es un algoritmo de procesamiento de lenguaje natural (NLP) que se utiliza para representar palabras como vectores numéricos en un espacio vectorial de alta dimensionalidad. Fue desarrollado por Mikolov et al. de Google en 2013 y ha sido una contribución importante al campo del NLP y la representación de palabras.

Word2Vec se basa en la idea de que las palabras que aparecen en contextos similares tienden a tener significados similares, y utiliza esta noción para aprender representaciones vectoriales de palabras a partir de grandes cantidades de texto sin etiquetar.

Aquí hay dos enfoques comunes para entrenar modelos Word2Vec:

1. **Skip-gram**: En este enfoque, se trata de predecir las palabras circundantes (contexto) a partir de una palabra central. Por ejemplo, si tienes la oración "El gato se sienta en el sofá", el modelo Skip-gram podría aprender a predecir "el", "gato", "se", "sienta", "en", "el", y "sofá" a partir de la palabra central "se". Esto ayuda a capturar las relaciones entre palabras en un contexto específico.
2. **CBOW (Continuous Bag of Words)**: En este enfoque, el modelo intenta predecir la palabra central a partir de las palabras circundantes. Usando el mismo ejemplo, el modelo CBOW intentaría predecir "se" a partir de las palabras circundantes "El", "gato", "sienta", "en", "el", y "sofá". Este enfoque es más rápido en términos de entrenamiento y se suele utilizar cuando se dispone de grandes cantidades de datos.

El proceso de entrenamiento de Word2Vec implica aprender representaciones vectoriales de palabras de manera que palabras similares estén cerca en el espacio vectorial. Estos vectores numéricos capturan propiedades semánticas y relaciones entre palabras. Por ejemplo, si representas "rey" y "reina" como vectores, la dirección entre los dos vectores podría representar la relación de género ("reina" - "rey" es similar a "mujer" - "hombre").

Word2Vec ha demostrado ser útil en una variedad de tareas de procesamiento de lenguaje natural, como la búsqueda de documentos, la traducción automática, la clasificación de texto y la agrupación de documentos, entre otros. Las representaciones vectoriales de palabras aprendidas por Word2Vec a menudo se utilizan como características de entrada para modelos de aprendizaje automático en tareas NLP más avanzadas. Estas representaciones ayudan a mejorar la eficacia de los modelos al capturar mejor el significado y las relaciones entre las palabras en los datos de texto.

A continuación se presenta el desarrollo de un conjunto de pruebas a realizar para reforzar el funcionamiento de los algoritmos antes mencionados.

B) Procedimiento:

1.-Obtener los conjuntos de pruebas del repositorio proporcionado.

2.-Desplegar la información si se desea para observar el contenido de los archivos dados.

3.-Se proporcionan los archivos base (código) para ejecutar los ejercicios propuestos. Se encuentran al final en el apéndice A.

Actividades a realizar:

I) Se plantean dos formas de vectorizar un conjunto de palabras con respecto a las relaciones que se presenten en un documento de referencia (vocabulario), el primero es a través de la herramienta gemsim y el otro método es a través en general de una matriz de co-ocurrencia y la evaluación por medio de la similitud de las palabras (los códigos se proporcionan en el apendice A).

Modifique los códigos de tal forma que previo a vectorizar el conjunto de palabras:

- Elimine las stopwords del documento original
- Elimine las stopwords y símbolos especiales (.,$#%) del documento original

Observe y analice el rendimiento, planteé conclusiones de los resultados obtenidos. Describa además el funcionamiento del código en los puntos importantes en cuanto a la vectorización de las palabras.

Puede modificar el código base como usted lo desee.

II) Se plantean dos algoritmos (códigos apéndice A) para la clasificación de reseñas de películas como positivas o negativas. Observe el funcionamiento de estos dos algoritmos y describa el funcionamiento y explique por qué se obtiene el rendimiento en las pruebas que realice y plantee posibles soluciones de mejora, de ser posible coloque dichas mejoras agregando en el código base.

Puede modificar el código base como usted lo desee.

Una vez realizado las dos actividades propuestas, complete los apartados siguientes:

C) Resultados

D) Análisis de Resultados

E) Conclusiones

F) Referencias bibliográficas

Apéndice A

Códigos base a implementar

Código 1

```
import nltk

nltk.download('punkt')

import gensim

import numpy as np

import pandas as pd

from nltk.tokenize import word_tokenize

from gensim.test.utils import common_texts

from gensim.models import Word2Vec

with                                                                    open
('/content/drive/MyDrive/IPN_2023_Oton˜o_Presentaciones/reglamento_tran
sito.txt',

          'r',encoding='utf-8') as file:

       document = file.read()

document[:1000]

# tokenizar el documento en oraciones

sentences = nltk.sent_tokenize(document)

sentences[:10]

#tokenizar cada oracion en palabras

word_tokens    =    [nltk.tokenize.word_tokenize(sentence.lower())    for
sentence in sentences]

#Modelo word2vec

modelo_w2v=gensim.models.Word2Vec(sentences=word_tokens,vector_size=32,
window= 10,min_count=1, workers=4)

modelo_w2v.save("word2vec.model")
```

```
model = Word2Vec.load("word2vec.model")

model.train(sentences, total_examples=1, epochs=100)



sims = model.wv.most_similar('auto', topn=10)

print(type(sims))

df = pd. DataFrame (sims). T

print(df)

#------------Código 2-----------------------------------------------

import numpy as np



# Calcular similitud de palabras

def cosine_similarity(vec1, vec2):

   dot_product = np.dot(vec1, vec2)

   norm1 = np.linalg.norm(vec1)

   norm2 = np.linalg.norm(vec2)

   similarity = dot_product / (norm1 * norm2)

   return similarity



# Leer el conjunto de datos desde un archivo de texto plano

with
open('/content/drive/MyDrive/IPN_2023_Oton˜o_Presentaciones/reglamento_
transito.txt', 'r', encoding='utf-8') as file:

   sentences = [line.strip().split() for line in file]



# Crear un vocabulario único de palabras

vocab = list(set(word for sentence in sentences for word in sentence))
```

```
vocab.sort()


# Crear una matriz de co-ocurrencia

window_size = 2

co_occurrence_matrix = np.zeros((len(vocab), len(vocab)))


for sentence in sentences:

   for i, target_word in enumerate(sentence):

       target_index = vocab.index(target_word)

            context_window = sentence[max(0, i - window_size):i] +
sentence[i + 1:i + window_size + 1]


       for context_word in context_window:

           context_index = vocab.index(context_word)

           co_occurrence_matrix[target_index][context_index] += 1


# Aplicar SVD para obtener vectores de palabras

U, S, Vt = np.linalg.svd(co_occurrence_matrix)


# Reducir la dimensionalidad (opcional)

vector_size = 100

word_vectors = U[:, :vector_size]


# Consultar el vector de una palabra específica

target_word = "peatón"
```

```
target_index = vocab.index(target_word)

vector = word_vectors[target_index]

print(f"Vector de '{target_word}':", vector)

similar_words = {}

for word in vocab:

   word_index = vocab.index(word)

         similarity   =   cosine_similarity(word_vectors[target_index],
word_vectors[word_index])

   similar_words[word] = similarity



sorted_similar_words   =   sorted(similar_words.items(),   key=lambda   x:
x[1], reverse=True)

top_similar_words = sorted_similar_words[:3]

print(f"Palabras similares a '{target_word}': {top_similar_words}")



#------------- Código 3 ------------------------------

import numpy as np

import pandas as pd

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import train_test_split

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report



# Cargar el conjunto de datos

data                                                                =
pd.read_csv("/content/drive/MyDrive/IPN_2023_Oton˜o_Presentaciones/NLP/
```

```
movie_review.csv")  # Asegúrate de tener un archivo CSV con reseñas y etiquetas (positivas/negativas).


# Preprocesamiento de datos

# Supongamos que tienes una columna 'texto' con las reseñas y una columna 'etiqueta' con las etiquetas (0 para negativo, 1 para positivo).


# Separar los datos en características (X) y etiquetas (y)

X = data['text']

y = data['tag']


# Dividir los datos en conjuntos de entrenamiento y prueba

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Paso 4: Vectorización de texto

vectorizer = TfidfVectorizer(max_features=5000)  # Convertir el texto en características numéricas (TF-IDF)

X_train = vectorizer.fit_transform(X_train)

X_test = vectorizer.transform(X_test)


# Paso 5: Entrenar un modelo de clasificación

clf = MultinomialNB()

clf.fit(X_train, y_train)


# Paso 6: Realizar predicciones en el conjunto de prueba
```

```
y_pred = clf.predict(X_test)


# Paso 7: Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)


print(f"Precisión del modelo: {accuracy}")
print("Informe de clasificación:")
print(report)
#------------------------- Código 4 -------------
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.decomposition import PCA  # Importar PCA
import matplotlib.pyplot as plt


# Cargar el conjunto de datos (igual que en el ejemplo anterior)
data = pd.read_csv("movie_reviews.csv")
X = data['text']
y = data['tag']


# Dividir los datos en conjuntos de entrenamiento y prueba
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



vectorizer = TfidfVectorizer(max_features=5000)

X_train = vectorizer.fit_transform(X_train)

X_test = vectorizer.transform(X_test)


# Entrenar un modelo de clasificación (usar K-Nearest Neighbors en lugar de Naive Bayes)

k = 5  # Número de vecinos cercanos a considerar (puedes ajustarlo según tus necesidades)

clf = KNeighborsClassifier(n_neighbors=k)

clf.fit(X_train, y_train)


#  Reducción de dimensionalidad con PCA (para visualización)

pca = PCA(n_components=2)  # Reducir a solo 2 dimensiones para la visualización

X_train_pca = pca.fit_transform(X_train.toarray())


# Realizar predicciones en el conjunto de prueba

y_pred = clf.predict(X_test)


# Visualización de la agrupación en un gráfico de dispersión

plt.figure(figsize=(10, 6))

colors = ['b', 'r']  # Azul para etiqueta 0 (negativo) y rojo para etiqueta 1 (positivo)
```

```
for label in np.unique(y_train):

   indices = np.where(y_train == label)

       plt.scatter(X_train_pca[indices,  0],  X_train_pca[indices,  1],
c=colors[label], label=label)



plt.title('Agrupación de Reseñas de Películas (PCA)')

plt.xlabel('Dimensión 1')

plt.ylabel('Dimensión 2')

plt.legend(['Negativo', 'Positivo'])

plt.show()
```