INSTITUTO TECNOLÓGICO BELTRÁN
Centro de Tecnología e Innovación

Modelos probabilísticos: Basados en N-Gramas

Alumno: Marco Medina
Fechas: 21/06
Institución: Instituto Tecnologico Beltran

En esta práctica vamos a usar de las técnicas más utilizadas en el procesamiento del lenguaje natural (PLN), los N-Gramas nos permite estudiar la distribución de secuencias de palabras en un texto, útil para modelado de lenguaje, predicción de palabras y análisis de contenido.

Para desarrollar esta técnica en nuestro fue necesario de 3 cosas:

- Un corpus a analizar
- las BIBLIOTECAS con las que limpiaremos y analizaremos el corpus
- Tener en mente que tipo de N-grama vamos a graficar su frecuencia

El análisis fue implementado en Python utilizando bibliotecas especializadas como `NLTK`, `sklearn`, `pandas` y `seaborn`. El código se estructuró modularmente a través de funciones.

Para poder usar estas bibliotecas hay que descargarlas en el cuerpo del código y la terminal:

- **En el cuerpo:**
  nltk.download("stopwords")
  nltk.download("punkt")
  nltk.download("wordnet")
  nltk.download("averaged_perceptron_tagger")
- **En la terminal:**
  pip install nltk
  pip install pandas
  pip install seaborn
  pip install matplotlib

Se aplicaron los siguientes pasos sobre cada línea del corpus:

- **Tokenización**: segmentación de las líneas en palabras.
- **Eliminación de stopwords**: se eliminaron palabras vacías en español usando NLTK.
- **Lematización**: se transformaron las palabras a su forma base (por ejemplo, “estudiando” a “estudiar”).

```python
def quitar_stopwords_esp(texto):
    espaniol = stopwords.words("spanish")
    return [w.lower() for w in texto if w.lower() not in espaniol
            and w not in string.punctuation
            and w not in ["'s", '|', '--', "''", "``", "-", ",-", "2025"]]

def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

def lematizar(texto):
    lemmatizer = WordNetLemmatizer()
    return [lemmatizer.lemmatize(w, get_wordnet_pos(w)) for w in texto]

def preparar_corpus(ruta_archivo, nombre_archivo):
    corpus = PlaintextCorpusReader(ruta_archivo, nombre_archivo, encoding="latin1")
    lineas = corpus.raw().splitlines()
    corpus_final = []

    for linea in lineas:
        tokens = word_tokenize(linea)
        limpio = quitar_stopwords_esp(tokens)
        lema = lematizar(limpio)
        if lema:
            corpus_final.append(" ".join(lema))
    return corpus_final
```

Luego de preparar el corpus, generamos los N-Gramas. Se utilizó la clase `CountVectorizer` de `sklearn` para generar:

- **Bigramas**: combinaciones de dos palabras contiguas.
- **Trigramas**: combinaciones de tres palabras contiguas.

Para evitar ruido, se definió un parámetro `min_df=2`, lo cual indica que solo se consideran las secuencias que aparecen al menos en dos documentos (lineas).

```python
def generar_ngrama(texto, rango_ngramas=(2, 3), min_df=2):
    vectorizer = CountVectorizer(ngram_range=rango_ngramas, min_df=min_df)
    X = vectorizer.fit_transform(texto)
    suma = X.sum(axis=0)
    frecs = [(ngram, suma[0, idx]) for ngram, idx in vectorizer.vocabulary_.items()]
    frecs_ordenadas = sorted(frecs, key=lambda x: x[1], reverse=True)
    return frecs_ordenadas
```

Y finalmente nos queda realizar la concatenación de los gráficos, en el cual se compara la frecuencia de bi-gramas y tri-gramas:

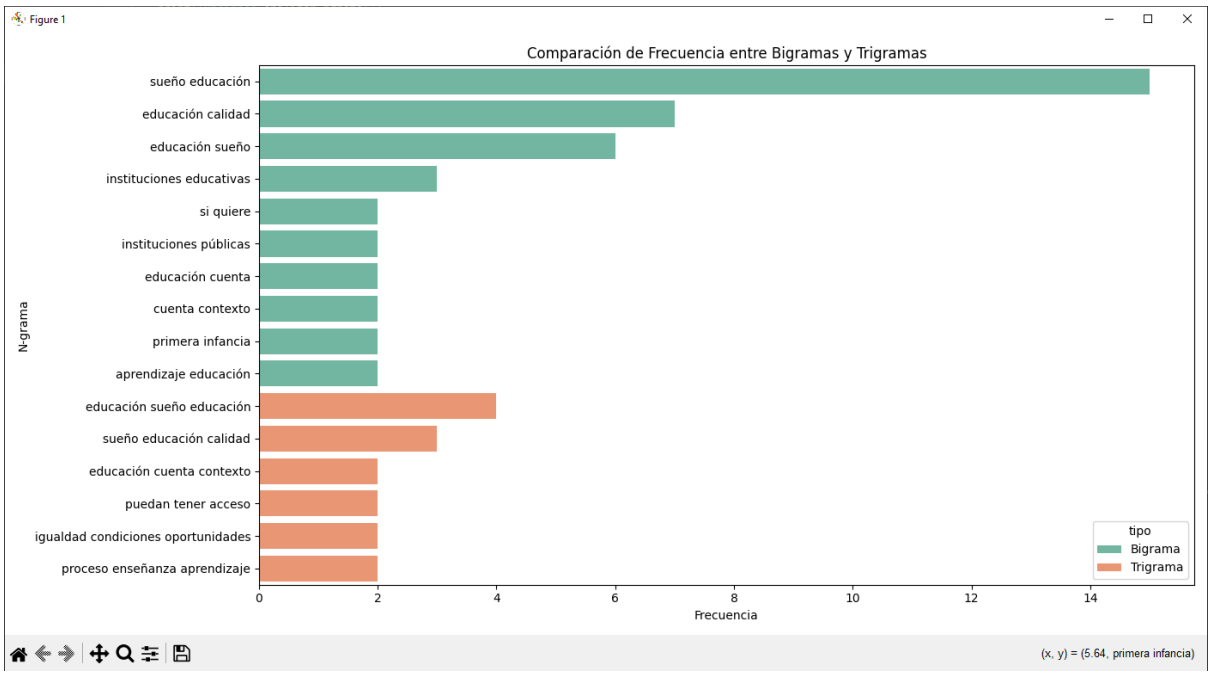
```python
def graficar_comparacion(df_bi, df_tri):
    df_comparacion = pd.concat([df_bi, df_tri])
    plt.figure(figsize=(14, 7))
    sns.barplot(data=df_comparacion, x="frecuencia", y="ngram", hue="tipo", palette="Set2")
    plt.title("Comparación de Frecuencia entre Bigramas y Trigramas")
    plt.xlabel("Frecuencia")
    plt.ylabel("N-grama")
    plt.tight_layout()
    plt.show()
```

Resultado:

- Se llama a la función **analizar_ngrama:**

```python
def analizar_ngrams(ruta_archivo=".", nombre_archivo="CorpusEducacion.txt", min_df=2):
    corpus_final = preparar_corpus(ruta_archivo, nombre_archivo)

    frecs_bi = generar_ngrama(corpus_final, rango_ngramas=(2, 2), min_df=2)
    frecs_tri = generar_ngrama(corpus_final, rango_ngramas=(3, 3), min_df=min_df)

    df_bi = pd.DataFrame(frecs_bi, columns=["ngram", "frecuencia"]).head(10)
    df_bi["tipo"] = "Bigrama"

    df_tri = pd.DataFrame(frecs_tri, columns=["ngram", "frecuencia"]).head(10)
    df_tri["tipo"] = "Trigrama"

    graficar_comparacion(df_bi, df_tri)
```

- Desde nuestro **main** printeamos la función:



Conclusiones

- Los modelos N-grama permiten descubrir estructuras de lenguaje frecuentes en corpus textuales.

- La limpieza del texto (stopwords, lematización) es esencial para eliminar ruido lingüístico y obtener combinaciones más significativas.

- Este enfoque puede ser útil en sistemas de recomendación, motores de búsqueda, o análisis de sentimientos.