Python Feature Engineering - Capítulo 11: "Extracting Features from Text Variables"

Soledad Galli

October 28, 2025

Resumen Completo del Capítulo 11: Extrayendo Características de Variables de Texto

Este capítulo se enfoca en las técnicas de **Procesamiento de Lenguaje Natural (NLP)** para transformar datos de texto no estructurados en características numéricas y estructuradas que puedan ser utilizadas por modelos de machine learning. A diferencia de los datos tabulares, el texto varía en longitud, contenido y estilo, pero contiene información valiosa que puede ser extraída.

El capítulo utiliza principalmente las librerías de Python: **pandas**, **scikit-learn** y **NLTK (Natural Language Toolkit)**.

Requisitos Técnicos Iniciales

Antes de comenzar, el capítulo indica que se deben instalar las librerías y descargar los paquetes necesarios de NLTK para la tokenización y las palabras vacías (*stop words*).

```
# Comando a ejecutar en una consola de Python tras instalar NLTK
import nltk

nltk.download('punkt')
nltk.download('stopwords')
```

Este capítulo cuenta con cinco "recetas". Las vemos a continuación...

---

1. Conteo de Características Estadísticas (Caracteres, Palabras y Vocabulario)

Esta técnica busca capturar la complejidad de un texto a través de métricas estadísticas simples, sin necesidad de interpretar su contenido.

- **Objetivo**: Inferir la cantidad de información de un texto. Textos más largos, con más palabras únicas y palabras más largas, suelen ser más ricos en detalles.

- **Características Extraídas**:

    1. **Número total de caracteres** (`num_char`).

    2. **Número total de palabras** (`num_words`).

    3. **Número de palabras únicas (vocabulario)** (`num_vocab`).

    4. **Diversidad léxica** (`lexical_div`): `num_words` / `num_vocab`.

5. **Longitud promedio de palabra** (`ave_word_length`): `num_char` / `num_words`.

- **Herramientas**: Se utiliza `pandas` y sus funciones vectorizadas de strings (`.str`).

Código de Implementación

```
# 1. Cargar las librerías y el dataset
import pandas as pd
from sklearn.datasets import fetch_20newsgroups

# Cargar la porción de entrenamiento del dataset
data = fetch_20newsgroups(subset='train')
df = pd.DataFrame(data.data, columns=['text'])

# 2. Calcular el número de caracteres
df['num_char'] = df['text'].str.len()

# 3. Calcular el número de palabras
df['num_words'] = df['text'].str.split().str.len()

# 4. Calcular el número de palabras únicas (vocabulario)
df['num_vocab'] =
↪  df['text'].str.lower().str.split().apply(set).str.len()

# 5. Calcular la diversidad léxica
df['lexical_div'] = df['num_words'] / df['num_vocab']

# 6. Calcular la longitud promedio de las palabras
df['ave_word_length'] = df['num_char'] / df['num_words']

# Visualizar las primeras 5 filas con las nuevas características
print(df.head())
```

	text	num_char	num_words	num_vocab	lexical_div	ave_word_length
0	From: lerxst@wam.umd.edu (where's my thing)\nS...	716	123	93	1.322581	5.821138
1	From: guykuo@carson.u.washington.edu (Guy Kuo)...	857	123	99	1.242424	6.967480
2	From: twillis@ec.ecn.purdue.edu (Thomas E Will...	1980	339	219	1.547945	5.840708
3	From: jgreen@amber (Joe Green)\nSubject: Re: W...	814	113	96	1.177083	7.203540
4	From: jcm@head-cfa.harvard.edu (Jonathan McDow...	1117	171	139	1.230216	6.532164

Figura 11.1 - Un DataFrame con la variable de texto y características que resumen algunas de las propiedades del texto.

Código para Visualización

Estas características pueden ser útiles para diferenciar categorías. El siguiente código visualiza la distribución del número de palabras para cada uno de los 20 temas del dataset.

```
import matplotlib.pyplot as plt

# Añadir la columna de 'target' (el tema de la noticia) al DataFrame
df['target'] = data.target

# Función para graficar histogramas de una característica para cada tema
def plot_features(df, text_var):
    nb_rows = 5
    nb_cols = 4
    fig, axs = plt.subplots(nb_rows, nb_cols, figsize=(12, 12))
    plt.subplots_adjust(wspace=None, hspace=0.4)
    n = 0
    for i in range(0, nb_rows):
        for j in range(0, nb_cols):
            axs[i, j].hist(df[df.target == n][text_var], bins=30)
            axs[i, j].set_title(text_var + ' | ' + str(n))
            n += 1
    plt.show()

# Ejecutar la función para la característica 'num_words'
plot_features(df, 'num_words')
```

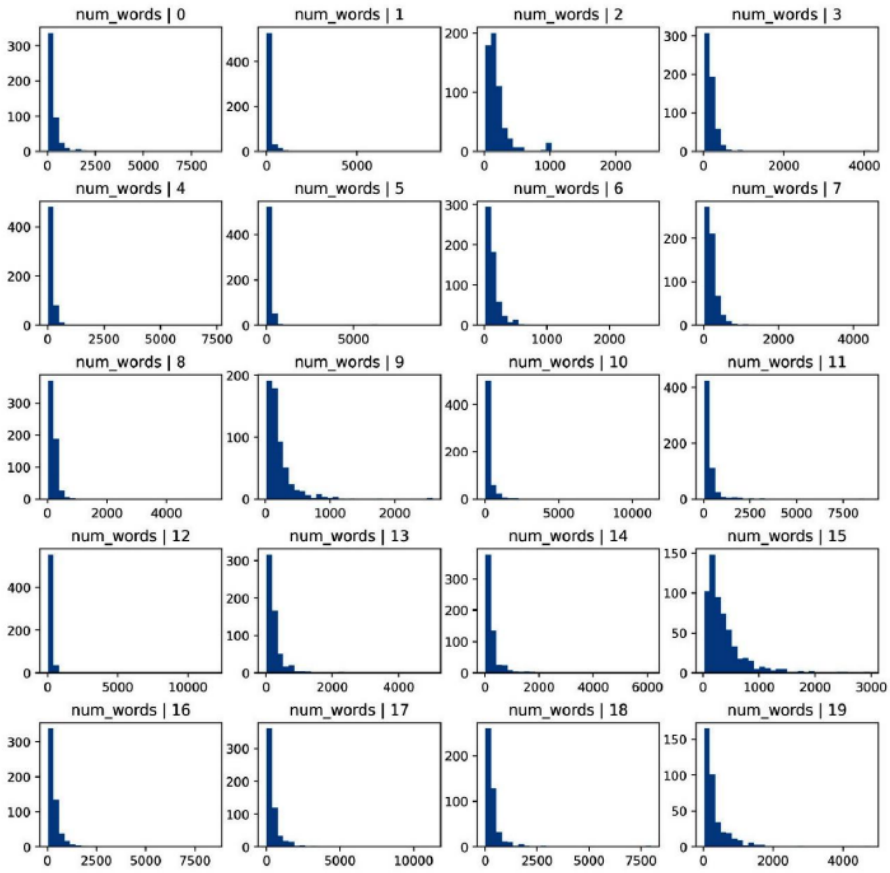

Figura 11.2 - Histogramas que muestran la distribución del número de palabras por texto, segregados por el tema discutido en cada texto.

---

2. Estimación de la Complejidad del Texto Contando Oraciones

- **Concepto Clave**: **Tokenización de oraciones (Sentence Tokenization)**. Es el proceso de dividir un bloque de texto en una lista de oraciones individuales.

- **Herramientas**: Se utiliza la función `sent_tokenize` de la librería **NLTK**.

Código de Implementación

```
# 1. Cargar las librerías necesarias
import pandas as pd
from nltk.tokenize import sent_tokenize
```

```
from sklearn.datasets import fetch_20newsgroups

# Ejemplo con un string simple para entender el tokenizador
text = """The alarm rang at 7 in the morning as it usually did on
↪  Tuesdays. She rolled over,
stretched her arm, and stumbled to the button till she finally managed
↪  to switch it
off. Reluctantly, she got up and went for a shower."""
sentences = sent_tokenize(text)
print(f"Oraciones encontradas: {len(sentences)}")

# 2. Aplicación a un DataFrame
data = fetch_20newsgroups(subset='train')
df = pd.DataFrame(data.data, columns=['text'])

# Para acelerar, trabajar solo con las primeras 10 filas
df = df.loc[1:10]

# Eliminar cabeceras de email (todo lo que viene antes de 'Lines:')
df['text'] = df['text'].str.split('Lines:').apply(lambda x: x[1] if
↪  len(x) > 1 else x[0])

# Crear la nueva característica 'num_sent' aplicando el tokenizador a
↪  cada texto
df['num_sent'] = df['text'].apply(sent_tokenize).apply(len)

# Visualizar el resultado
print(df[['text', 'num_sent']])
```

	text	num_sent
1	11\nNNTP-Posting-Host: carson.u.washington.ed...	6
2	36\n\nwell folks, my mac plus finally gave up...	9
3	14\nDistribution: world\nNNTP-Posting-Host: a...	7
4	23\n\nFrom article <C5owCB.n3p@world.std.com>...	10
5	58\n\nIn article <1r1eu1$4t@transfer.stratus....	21
6	12\n\nThere were a few people who responded t...	8
7	44\nDistribution: world\nNNTP-Posting-Host: d...	15
8	10\n\nI have win 3.0 and downloaded several i...	3
9	29\n\njap10@po.CWRU.Edu (Joseph A. Pellettier...	12
10	13\n\nI have a line on a Ducati 900GTS 1978 m...	11

Figura 11.3 - Un DataFrame con la variable de texto y el número de oraciones por texto.

---

3. Creación de Características con Bag-of-Words (BoW) y N-gramas

Esta es una de las técnicas más fundamentales para convertir texto en vectores numéricos.

- **Concepto Clave: Bag-of-Words (BoW)**. Representación que describe un texto contando la frecuencia de cada palabra, ignorando por completo el orden y la gramática.

- **Concepto Clave: N-gramas**. Secuencias contiguas de n palabras que capturan parte del contexto que el BoW simple pierde.

- **Herramientas**: Se utiliza `CountVectorizer` de **scikit-learn**.

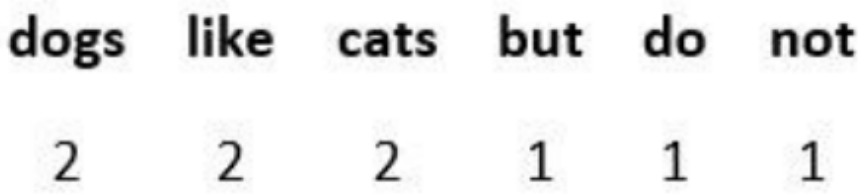

Figura 11.4 - El BoW derivado de la oración “Dogs like cats, but cats do not like dogs”.

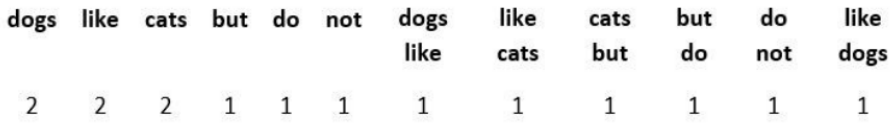

Figura 11.5 - El BoW con 2-gramas.

Código de Implementación

```
# 1. Cargar librerías y datos
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer

data = fetch_20newsgroups(subset='train')
df = pd.DataFrame(data.data, columns=['text'])

# 2. Limpieza básica: eliminar puntuación y números
df['text'] = df['text'].str.replace(r'[^\w\s]', '',
↪  regex=True).str.replace(r'\d+', '', regex=True)

# 3. Configurar CountVectorizer
vectorizer = CountVectorizer(
    lowercase=True,
    stop_words='english',
    ngram_range=(1, 1), # Se puede cambiar a (1, 2) para incluir
↪  bigramas
    min_df=0.05 # Ignorar términos que aparecen en menos del 5% de los
↪  documentos
)

# 4. Aprender el vocabulario y transformar el texto
vectorizer.fit(df['text'])
X = vectorizer.transform(df['text'])
```

```python
# 5. Crear un DataFrame con los resultados
bagofwords = pd.DataFrame(
    X.toarray(),
    columns=vectorizer.get_feature_names_out()
)

# Visualizar las primeras filas de la matriz BoW
print(bagofwords.head())
```

	able	access	actually	ago	apr	article	articleid	ask	available	away	...	works	world	writes	wrong	wrote	xnewsreader	year	years	yes	youre
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	1	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	1	0	0	0	0	...	0	1	1	0	1	1	0	0	0	0
4	0	0	0	0	0	2	0	0	0	0	...	0	0	1	0	0	0	0	0	1	0

5 rows × 191 columns

Figura 11.6 - Un DataFrame con el BoW resultante del conjunto de datos 20 Newsgroup.

---

4. Implementación de TF-IDF (Frecuencia de Término–Frecuencia Inversa de Documento)

TF-IDF es una mejora sobre el BoW que asigna un "peso" o "importancia" a cada palabra, dando más valor a los términos que son frecuentes en un documento pero raros en el resto del corpus.

- **Herramientas**: Se utiliza `TfidfVectorizer` de **scikit-learn**.

Código de Implementación

```python
# 1. Cargar librerías y datos
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer

data = fetch_20newsgroups(subset='train')
df = pd.DataFrame(data.data, columns=['text'])

# 2. Limpieza básica (igual que en BoW)
df['text'] = df['text'].str.replace(r'[^\w\s]', '',
↪  regex=True).str.replace(r'\d+', '', regex=True)

# 3. Configurar TfidfVectorizer (parámetros similares a CountVectorizer)
```

```python
vectorizer = TfidfVectorizer(
    lowercase=True,
    stop_words='english',
    ngram_range=(1, 1),
    min_df=0.05
)

# 4. Aprender el vocabulario, calcular IDF y transformar los datos
vectorizer.fit(df['text'])
X = vectorizer.transform(df['text'])

# 5. Crear un DataFrame con los resultados
tfidf = pd.DataFrame(
    X.toarray(),
    columns=vectorizer.get_feature_names_out()
)

# Visualizar las primeras filas de la matriz TF-IDF
print(tfidf.head())
```

	able	access	actually	ago	apr	article	articleid	ask	available	away	...
0	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...
1	0.0	0.000000	0.000000	0.0	0.0	0.000000	0.356469	0.0	0.0	0.0	...
2	0.0	0.135765	0.123914	0.0	0.0	0.000000	0.000000	0.0	0.0	0.0	...
3	0.0	0.000000	0.000000	0.0	0.0	0.110035	0.000000	0.0	0.0	0.0	...
4	0.0	0.000000	0.000000	0.0	0.0	0.262692	0.000000	0.0	0.0	0.0	...

works	world	writes	wrong	wrote	xnewsreader	year	years	yes	youre
0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.27302	0.000000	0.0
0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.00000	0.000000	0.0
0.0	0.000000	0.000000	0.0	0.000000	0.000000	0.0	0.00000	0.000000	0.0
0.0	0.169635	0.100554	0.0	0.218197	0.233578	0.0	0.00000	0.000000	0.0
0.0	0.000000	0.120029	0.0	0.000000	0.000000	0.0	0.00000	0.264836	0.0

Figura 11.7 - Un DataFrame con características resultantes de TF-IDF.

---

5. Limpieza y Derivación (Stemming) de Variables de Texto

Este es un paso de preprocesamiento crucial que se realiza *antes* de aplicar técnicas como BoW o TF-IDF para estandarizar el texto.

- **Objetivo**: Enfocarse en el "mensaje" del texto eliminando variaciones irrelevantes.

- **Pasos del Proceso**:

  1. Eliminación de puntuación y números.
  2. Conversión a minúsculas.
  3. Eliminación de "Stop Words".
  4. **Stemming (Derivación)**: Reducción de palabras a su raíz (ej. "playing" -> "play").

- **Herramientas**: **pandas**, `stopwords` y `SnowballStemmer` de **NLTK**.

Código de Implementación

```
# 1. Cargar librerías y datos
import pandas as pd
from sklearn.datasets import fetch_20newsgroups
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer

data = fetch_20newsgroups(subset='train')
df = pd.DataFrame(data.data, columns=['text'])

# 2. Eliminar puntuación y números
df['text'] = df['text'].str.replace(r'[^\w\s]', '', regex=True)
df['text'] = df['text'].str.replace(r'\d+', '', regex=True)

# 3. Convertir a minúsculas
df['text'] = df['text'].str.lower()

# 4. Eliminar "stop words"
stop = set(stopwords.words('english'))
def remove_stopwords(text):
    words = [word for word in text.split() if word not in stop]
    return " ".join(words)

df['text'] = df['text'].apply(remove_stopwords)

# 5. Aplicar Stemming
stemmer = SnowballStemmer("english")
def stemm_words(text):
    words = [stemmer.stem(word) for word in text.split()]
    return " ".join(words)
```

```
df['text'] = df['text'].apply(stemm_words)

# 6. Visualizar un ejemplo del texto completamente procesado
print("Ejemplo de texto limpiado y con stemming:")
print(df['text'][10])
```