Práctica 2.2: Vectorización y Sistema de Búsqueda de Libros

UNR - TUIA - Procesamiento de Lenguaje Natural

PARTE 1: Ejercicios Preparatorios Simples

Ejercicio 1: Vectorización Básica con Corpus Simple

```
# Corpus simple para practicar
sinopsis_ejemplo = [
    "Una historia de amor en tiempos de guerra",
    "Aventuras mágicas en un mundo fantástico",
    "Un detective resuelve misterios en la ciudad",
    "Romance juvenil en una escuela",
    "Ciencia ficción sobre viajes en el tiempo"
]

def ejercicio_vectorizacion_basica(sinopsis):
    """
    TODO: Implementar y comparar:
    1. Count Vectorizer básico
    2. TF-IDF básico
    3. Mostrar las matrices resultantes
    4. Calcular similitud entre la primera y segunda sinopsis
    """
    pass
```

Ejercicio 2: Word Embeddings Simple

```
def ejercicio_embeddings_simple(sinopsis):
    """
    TODO:
    1. Tokenizar las sinopsis
    2. Entrenar un modelo Word2Vec simple (o usar uno preentrenado)
    3. Encontrar palabras similares a: "amor", "magia", "detective"
    4. Calcular similitud entre embeddings de palabras
    """
    pass
```

PARTE 2: Análisis con Dataset Real

Ejercicio 3: Carga y Exploración del Dataset

```
def explorar_dataset():
    """
    TODO:
    1. Cargar lectulandia_books.csv
    2. Explorar la columna 'sinopsis':
       - Cantidad de libros
       - Longitud promedio de sinopsis
       - Palabras más frecuentes
       - Distribución de géneros
    3. Limpiar y preprocesar las sinopsis
    """
    pass
```

Ejercicio 4: Implementación de Vectorización Frecuentista

```
class VectorizadorFrecuentista:
    def __init__(self, metodo='tfidf'):
        """
        TODO: Implementar clase que soporte:
        - Count Vectorizer
        - TF-IDF
        - Configuración de parámetros (max_features, ngram_range, etc.)
        """
        pass

    def entrenar(self, textos):
        """Entrenar el vectorizador con las sinopsis"""
        pass

    def vectorizar(self, textos):
        """Convertir textos a vectores"""
        pass

    def buscar_similares(self, query, n_resultados=5):
        """Encontrar libros similares usando similitud de coseno"""
        pass
```

Ejercicio 5: Implementación de Vectorización Semántica

```python
class VectorizadorSemantico:
    def __init__(self, modelo='word2vec'):
        """
        TODO: Implementar clase que soporte:
        - Word2Vec (entrenado o preentrenado)
        - FastText
        - Promedio de embeddings para representar documentos
        """
        pass

    def entrenar(self, textos):
        """Entrenar o cargar modelo de embeddings"""
        pass

    def vectorizar_documento(self, texto):
        """Convertir un documento a vector (promedio de word
embeddings)"""
        pass

    def buscar_similares(self, query, n_resultados=5):
        """Encontrar libros similares usando similitud de coseno"""
        pass
```

PARTE 3: Sistema de Búsqueda Comparativo

Ejercicio 6: Sistema de Búsqueda Integrado

```python
class SistemaBusquedaLibros:
    def __init__(self, df_libros):
        """
        TODO: Inicializar con:
        - DataFrame de libros
        - Ambos vectorizadores
        - Índices para búsqueda rápida
        """
        self.df = df_libros
        self.vectorizador_freq = VectorizadorFrecuentista()
        self.vectorizador_sem = VectorizadorSemantico()
        pass

    def entrenar_modelos(self):
        """Entrenar ambos tipos de vectorizadores"""
        pass
```

```
    def buscar(self, query, metodo='ambos'):
        """
        TODO: Implementar búsqueda que retorne:
        - Top 5 libros por método frecuentista
        - Top 5 libros por método semántico
        - Top 5 libros por combinación de ambos (promedio de scores)

        Retornar: dict con resultados de cada método
        """
        pass

    def mostrar_resultados(self, resultados, query):
        """Mostrar resultados de forma clara y comparativa"""
        pass
```

Ejercicio 7: Evaluación y Comparación

```
def evaluar_sistema():
    """
    TODO: Crear evaluación sistemática:
    1. Definir queries de prueba variadas:
       - "historia de amor romántica"
       - "aventuras de fantasía épica"
       - "misterio y suspense policial"
       - "ciencia ficción futurista"
       - "drama familiar contemporáneo"

    2. Para cada query, comparar:
       - Tiempo de respuesta
       - Relevancia de resultados (análisis manual)
       - Diversidad de géneros en resultados
       - Cobertura de vocabulario

    3. Crear métricas de evaluación:
       - Precisión subjetiva (1-5)
       - Diversidad de resultados
       - Tiempo de ejecución
    """
    pass
```

PARTE 4: Análisis y Visualización

Ejercicio 8: Análisis Comparativo Detallado

```python
def analisis_comparativo():
    """
    TODO: Generar análisis completo:
    1. Matriz de similitud entre métodos
    2. Visualización t-SNE de embeddings de libros
    3. Análisis de clusters por género
    4. Casos donde cada método funciona mejor
    5. Limitaciones identificadas de cada enfoque
    """
    pass
```

Ejercicio 9: Interfaz de Usuario Simple

```python
def crear_interfaz_busqueda():
    """
    TODO: Crear interfaz simple (puede ser por consola) que:
    1. Permita ingresar una consulta
    2. Muestre resultados de ambos métodos lado a lado
    3. Incluya información del libro (título, autor, géneros)
    4. Permita al usuario calificar la relevancia
    5. Guarde estadísticas de uso
    """
    pass
```