SAMSUNG

Curso Introducción a la Inteligencia Artificial

Proyecto Capstone

Documentación del Modelo JO Search

Este informe documenta el código desarrollado para el proyecto del Samsung Innovation Campus (SIC) 2023, cuyo objetivo es mejorar la eficiencia en la búsqueda de convocatorias en la plataforma pública Secop II.

Introducción

El propósito del proyecto es crear un sistema que permita buscar convocatorias relevantes utilizando palabras clave específicas. El sistema debe:

- Actualizar y almacenar datos desde una API.
- Preprocesar el texto para la búsqueda.
- Utilizar TF-IDF para la vectorización y cálculo de similitudes.
- Formatear y presentar los resultados de las búsquedas.
- Generar y descargar archivos CSV con los resultados.

Tipo de Modelo: Modelo de Recuperación de Información (Information Retrieval)

Componentes Principales:

1. **Vectorización de Texto (TF-IDF):** Utiliza la técnica de Term Frequency-Inverse Document Frequency (TF-IDF) para convertir los textos de las descripciones de las convocatorias en vectores numéricos.
2. **Similitud de Coseno:** Utiliza la similitud de coseno para calcular la similitud entre el vector de la consulta del usuario y los vectores de las descripciones de las convocatorias.

Clasificación del Modelo

1. Preprocesamiento del Texto:

- **Lematización:** Uso de WordNetLemmatizer para reducir las palabras a su forma base.
- **Eliminación de Palabras Vacías:** Uso del conjunto de palabras vacías en español de NLTK para eliminar palabras comunes que no aportan valor semántico significativo.
- **Normalización de Texto:** Conversión a minúsculas y eliminación de caracteres no alfanuméricos y espacios en blanco adicionales.

2. Vectorización de Texto:

- **TF-IDF Vectorizer:** El modelo utiliza TfidfVectorizer de la biblioteca scikit-learn para convertir el texto preprocesado en vectores TF-IDF, los cuales representan la importancia de las palabras en el contexto de las convocatorias.

3. Cálculo de Similitud:

- **Similitud de Coseno:** El modelo calcula la similitud de coseno entre el vector de la consulta del usuario y los vectores de las convocatorias para identificar las convocatorias más relevantes.

4. Clasificación y Recuperación:

- **Identificación de Convocatorias Relevantes:** Se clasifican las convocatorias en función de la similitud de coseno y se recuperan las más relevantes para la consulta del usuario.

Descripción del Proceso

1. **Obtención de Datos:**
   - Se obtienen los datos de la API de Secop II y se almacenan en un archivo CSV para facilitar su acceso y manipulación.
2. **Preprocesamiento de Texto:**
   - Las descripciones de las convocatorias se preprocesan para eliminar ruido y normalizar el texto, mejorando así la calidad de los vectores TF-IDF generados.
3. **Vectorización de Descripciones:**
   - Las descripciones preprocesadas se convierten en vectores TF-IDF, que representan la importancia de las palabras en cada descripción de convocatoria.
4. **Búsqueda de Convocatorias:**
   - El usuario introduce una consulta de búsqueda que se preprocesa y se convierte en un vector TF-IDF.
   - Se calcula la similitud de coseno entre el vector de la consulta y los vectores de las descripciones de las convocatorias.
   - Se identifican y recuperan las convocatorias más relevantes según los puntajes de similitud.
5. **Presentación y Exportación de Resultados:**
   - Los resultados se formatean para una presentación clara y se generan en un archivo CSV que el usuario puede descargar.

Código y Documentación

```
# Instalar librerías necesarias
!pip install pandas requests scikit-learn nltk

# Importar librerías
```

```python
import requests
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
import re
import os
import csv

# Descargar datos NLTK si es necesario
nltk.download('stopwords')
nltk.download('wordnet')

# Definir la URL de la API y la ruta de almacenamiento de datos
api_url = "https://www.datos.gov.co/resource/p6dx-8zbt.json"
data_path = "secop_data.csv"

# Función para actualizar y almacenar datos desde la API
def update_data():
    response = requests.get(api_url)
    if response.status_code == 200:
        data = response.json()
        df = pd.DataFrame(data)
        df.to_csv(data_path, index=False)
        print("Data updated successfully!")
    else:
        print(f"Error accessing API: {response.status_code}")

# Verificar si los datos existen, de lo contrario, actualizarlos
if not os.path.exists(data_path):
    update_data()

# Cargar datos preprocesados
df = pd.read_csv(data_path)

# Definir función de preprocesamiento de texto
def preprocess_text(text):
    """
    Preprocesa el texto de entrada para la búsqueda.
    Args:
        text (str): Texto a preprocesar.
    Returns:
```

```
        str: Texto preprocesado.
    """
    text = text.lower()  # Convertir a minúsculas
    text = re.sub(r'[^\w\s]', '', text)  # Eliminar caracteres no
alfanuméricos
    text = re.sub(r'\s+', ' ', text)  # Eliminar espacios en blanco
adicionales
    tokens = text.split()  # Tokenizar el texto
    lemmatizer = WordNetLemmatizer()  # Inicializar lematizador
    tokens = [lemmatizer.lemmatize(token) for token in tokens]  #
Lematizar tokens
    stop_words = set(stopwords.words('spanish'))  # Obtener palabras
vacías en español
    tokens = [token for token in tokens if token not in stop_words]  #
Eliminar palabras vacías
    return ' '.join(tokens)  # Unir tokens preprocesados

# Preprocesar datos de descripciones y palabras clave
# Check if both columns exist before creating the 'processed_text'
column
if 'nombre_del_procedimiento' in df.columns and
'descripci_n_del_procedimiento' in df.columns:
    # Create the 'processed_text' column by combining and handling
missing values
    df['processed_text'] = df['nombre_del_procedimiento'].fillna('') +
' ' \
                           +
df['descripci_n_del_procedimiento'].fillna('')

    # Apply any preprocessing or analysis to the 'processed_text'
column as needed
    df['processed_text'] = df['processed_text'].apply(preprocess_text)
    print("LISTO")
else:
    # Handle the case where columns are missing
    print("Las columnas 'nombre_del_procedimiento' o
'descripci_n_del_procedimiento' no están disponibles. Se omitirá el
preprocesamiento de texto.")

from sklearn.feature_extraction.text import TfidfVectorizer

# Define keywords
keywords = [
    "Innovación", "Transformación digital", "Internacionalización",
```

```
    "Logística", "Analítica de datos", "Machine learning",
    "Emprendimiento", "Habilidades blandas", "Planeación estratégica",
    "Estudio de mercado", "Inteligencia de negocio", "Marketing",
"Comercio"
]
# Crear vectorizador TF-IDF
vectorizer = TfidfVectorizer(max_features=10000)  # Limitar el número
de características
vectorizer.fit(df['processed_text'])

# Definir función para buscar convocatorias
def search_tenders(query):
    """
    Busca convocatorias relevantes para una consulta dada.
    Args:
        query (str): Consulta de búsqueda.
    Returns:
        pandas.DataFrame: DataFrame con resultados de búsqueda.
    """
    query_vec = vectorizer.transform([preprocess_text(query)])
    similarity_scores = cosine_similarity(query_vec,
vectorizer.transform(df['processed_text'])).flatten()
    top_indices = similarity_scores.argsort()[-10:][::-1]
    results = df.iloc[top_indices].copy()
    results['puntuacion_similitud'] = similarity_scores[top_indices]
    return results

# Definir función para formatear la información de la convocatoria
def format_tender_info(results):
    """
    Formatea la información de las convocatorias encontradas.
    Args:
        results (pandas.DataFrame): DataFrame con resultados de
búsqueda.
    Returns:
        list: Lista de resultados formateados.
    """
    formatted_results = []
    for _, row in results.iterrows():
        tender_info = {
            "Fecha de publicación": row.get('fecha_publicacion'),
            "Entidad": row.get('entidad'),
            "NIT Entidad": row.get('nit_entidad'),
            "Departamento Entidad": row.get('departamento_entidad'),
```

```
            "Ciudad Entidad": row.get('ciudad_entidad'),
            "Nombre del Procedimiento":
row.get('nombre_del_procedimiento'),
            "Descripción del Procedimiento":
row.get('descripci_n_del_procedimiento'),
            "Precio Base": row.get('precio_base'),
            "Modalidad de Contratación":
row.get('modalidad_de_contratacion'),
            "Estado del Procedimiento":
row.get('estado_del_procedimiento'),
            "Valor Total Adjudicación":
row.get('valor_total_adjudicacion'),
            "Nombre del Adjudicador":
row.get('nombre_del_adjudicador'),
            "Nombre del Proveedor": row.get('nombre_del_proveedor'),
            "NIT Proveedor Adjudicado":
row.get('nit_del_proveedor_adjudicado'),
            "URL del Proceso": row.get('urlproceso')
        }
        formatted_result = "\n**Convocatoria:**\n"
        for key, value in tender_info.items():
            formatted_result += f"{key}: {value}\n"
        formatted_results.append(formatted_result)
    return formatted_results

# Definir función para generar CSV con los resultados formateados
def generate_csv(formatted_results, filename):
    """
    Genera un archivo CSV con los resultados formateados de la
búsqueda.
    Args:
        formatted_results (list): Lista de resultados formateados.
        filename (str): Nombre del archivo CSV a generar.
    """
    with open(filename, 'w', newline='') as csvfile:
        csv_writer = csv.writer(csvfile)
        header = ["Fecha de publicación", "Entidad", "NIT Entidad",
"Departamento Entidad",
                  "Ciudad Entidad", "Nombre del Procedimiento",
"Descripción del Procedimiento",
                  "Precio Base", "Modalidad de Contratación", "Estado
del Procedimiento",
                  "Valor Total Adjudicación", "Nombre del
Adjudicador", "Nombre del Proveedor",
```

```
                    "NIT Proveedor Adjudicado", "URL del Proceso"]
            csv_writer.writerow(header)
            for result in formatted_results:
                tender_info = result.split('\n')[1:]  # Extract tender
info from formatted result
                tender_data = [info.split(':')[1].strip() for info in
tender_info if ':' in info]  # Extract values
                csv_writer.writerow(tender_data)
    print(f"CSV generado: {filename}")

# Simular la descarga de un archivo CSV (funcionalidad proyectada)
def download_csv(filename):
    """
    Simula la descarga de un archivo CSV mostrando un enlace de
descarga (placeholder).
    Args:
        filename (str): Nombre del archivo CSV generado.
    """
    download_link = f"http://example.com/download/{filename}"  #
Placeholder link
    print(f"Haga clic aquí para descargar el CSV: {download_link}")

# Función principal
def main():
    # Verificar si los datos existen, de lo contrario, actualizarlos
    if not os.path.exists(data_path):
        update_data()

    # Cargar datos preprocesados
    df = pd.read_csv(data_path)

    # Obtener palabras clave del usuario y realizar búsqueda
    user_keywords = input("Ingrese sus palabras clave de búsqueda: ")
    results = search_tenders(user_keywords)
    formatted_results = format_tender_info(results)

    # Imprimir resultados formateados
    print("Resultados de búsqueda:")
    if not formatted_results:
        print("No se encontraron convocatorias que coincidan con sus
palabras clave.")
    else:
        for result in formatted_results:
            print(result)
```

```
            print("\n**¡Gracias por usar el modelo JO de Búsqueda
Rápida!**")

    # Generar y descargar CSV
    csv_filename = "resultados_convocatorias.csv"
    generate_csv(formatted_results, csv_filename)
    download_csv(csv_filename)

# Ejecutar función principal
if __name__ == "__main__":
    main()
```

Descripción de las Funciones y Módulos

1. **Instalación de Librerías**:
    - !pip install pandas requests scikit-learn nltk: Instala las librerías necesarias para la ejecución del código.
2. **Importación de Librerías**:
    - Importa las librerías requests, pandas, scikit-learn, nltk, re, os, y csv para la manipulación de datos, procesamiento de texto, y cálculos de similitud.
3. **Descarga de Datos NLTK**:
    - nltk.download('stopwords') y nltk.download('wordnet'): Descarga las palabras vacías y el lematizador de WordNet necesarios para el preprocesamiento de texto.
4. **Actualización y Almacenamiento de Datos**:
    - update_data(): Función para actualizar y almacenar los datos desde la API de Secop II en un archivo CSV.
5. **Verificación y Carga de Datos**:
    - Verifica si el archivo CSV existe. Si no, llama a update_data() para crear el archivo. Luego, carga los datos en un DataFrame de pandas.
6. **Preprocesamiento de Texto**:
    - preprocess_text(text): Función que convierte el texto a minúsculas, elimina caracteres no alfanuméricos y palabras vacías, y lematiza los tokens.

7. **Preprocesamiento de Descripciones**:
    - Verifica si las columnas necesarias existen en el DataFrame. Si existen, combina y preprocesa las columnas nombre_del_procedimiento y descripci_n_del_procedimiento.
8. **Vectorización TF-IDF**:
    - Crea un vectorizador TF-IDF y ajusta el modelo a los datos preprocesados.
9. **Búsqueda de Convocatorias**:
    - search_tenders(query): Busca convocatorias relevantes para una consulta dada utilizando la similitud coseno entre el vector de consulta y los vectores de las descripciones preprocesadas.
10. **Formateo de Información**:
    - format_tender_info(results): Formatea la información de las convocatorias encontradas en una lista de cadenas formateadas.
11. **Generación de CSV**:
    - generate_csv(formatted_results, filename): Genera un archivo CSV con los resultados formateados de la búsqueda.
12. **Simulación de Descarga de CSV**:
    - download_csv(filename): Simula la descarga de un archivo CSV mostrando un enlace de descarga placeholder.
13. **Función Principal**:
    - main(): Verifica si los datos existen, de lo contrario, los actualiza. Carga los datos preprocesados, obtiene palabras clave del usuario, realiza la búsqueda, formatea los resultados, y genera un archivo CSV con los resultados.

Ejecución del Código

- El código se ejecuta al llamar a la función main() cuando el archivo es ejecutado directamente.

Conclusión

El modelo propuesto se clasifica como un modelo de recuperación de información basado en técnicas de vectorización TF-IDF y cálculo de similitud de coseno. Este modelo permite identificar y recuperar las convocatorias más relevantes en la plataforma Secop II, mejorando la eficiencia y precisión de la búsqueda de convocatorias para los usuarios.

El tiempo de procesamiento promedio del modelo es entre 5 y 6 segundos, teniendo en cuenta que por limitación de la conexión API expuesta por la fuente solo podemos obtener las primeras 1000 filas.