Documentación de Prueba Técnica 2

Prueba

Construye un modelo básico de preguntas y respuestas (desde cero, no se aceptaran modelos pre-entrenados). En el código deben ser claras las fases de pre procesamiento, entrenamiento, validación y pruebas. La selección de los datos de entrada es libre. En tu script debes indicar claramente la fuente de datos que estas utilizando.

Solución

Se buscó por todos los medios públicos un dataset de preguntas y respuestas, así que se optó por utilizar el dataset público de Stanford:

https://rajpurkar.github.io/SQuAD-explorer/

Importar las bibliotecas necesarias:

```
import wikipedia as wiki
import json
import numpy as np
import pandas as pd
import os
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors
```

Se importan las bibliotecas necesarias, incluyendo wikipedia para realizar búsquedas en Wikipedia, json para trabajar con archivos JSON, numpy y pandas para manipular datos, os para manipulación de archivos y directorios, y TfidfVectorizer y NearestNeighbors de scikit-learn para implementar la recuperación de información basada en TF-IDF y búsqueda de vecinos más cercanos.

Establecer variables iniciales:

k = 5

question = "What are the tourist hotspots in Mexico?"

Se establece el valor de k como 5, que representa el número de resultados a obtener de una búsqueda en Wikipedia. La variable question contiene la pregunta que se utilizará para buscar información en Wikipedia.

Realizar la búsqueda en Wikipedia:

results = wiki.search(question, results=k)

print('Question:', question)

print('Pages: ', results)

Se utiliza la función search de wikipedia para buscar en Wikipedia los resultados relacionados con la pregunta. Se especifica el número de resultados deseado (results=k). Luego, se imprime en pantalla la pregunta y los resultados obtenidos.

Listar los datos disponibles en el directorio:

```
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

Este código recorre los directorios y archivos dentro de la ruta especificada (/kaggle/input) y muestra en pantalla la ruta completa de cada archivo encontrado.

Definir una función para convertir archivos JSON a un DataFrame:

```
def squad_json_to_dataframe(file_path, record_path=['data', 'paragraphs', 'qas', 'answers']):
    # Cargar el archivo JSON
```

```
    archivo = json.loads(open(file_path).read())

    # Analizar los diferentes niveles del archivo JSON
    js = pd.json_normalize(archivo, record_path)
    m = pd.json_normalize(archivo, record_path[:-1])
    r = pd.json_normalize(archivo, record_path[:-2])

    # Combinar todo en un solo dataframe
    idx = np.repeat(r['context'].values, r.qas.str.len())
    m['context'] = idx
    data = m[['id', 'question', 'context', 'answers']].set_index('id').reset_index()
    data['c_id'] = data['context'].factorize()[0]

    return data
```

Esta función toma la ruta de un archivo JSON y el record_path (ruta hasta el nivel más profundo en el archivo JSON) como argumentos. Carga el archivo JSON y lo analiza en diferentes niveles utilizando json_normalize de pandas. Luego, combina los resultados en un solo DataFrame y asigna un identificador único (c_id) a cada contexto. Finalmente, devuelve el DataFrame resultante.

Cargar los datos:

```
file_path = '/kaggle/input/stanford-question-answering-dataset/train-v1.1.json'
data = squad_json_to_dataframe(file_path)
```

Se especifica la ruta de un archivo JSON y se utiliza la función squad_json_to_dataframe para cargar

Contar la cantidad de datos únicos:

data['c_id'].unique().size

Se obtiene la cantidad de valores únicos en la columna 'c_id' del DataFrame data utilizando el método unique() y luego se obtiene el tamaño del resultado con el atributo size.

Crear un DataFrame de documentos únicos:

documents = data[['context', 'c_id']].drop_duplicates().reset_index(drop=True)

Se crea un nuevo DataFrame llamado documents que contiene las columnas 'context' y 'c_id' del DataFrame data, pero solo se mantienen las filas únicas utilizando el método drop_duplicates(). Luego se reinicia el índice del DataFrame resultante utilizando reset_index(drop=True).

Definir la configuración para TF-IDF y la recuperación de información:

```
tfidf_configs = {
   'lowercase': True,
   'analyzer': 'word',
   'stop_words': 'english',
   'binary': True,
   'max_df': 0.9,
   'max_features': 10_000
}
retriever_configs = {
   'n_neighbors': 10,
   'metric': 'cosine'
}
embedding = TfidfVectorizer(**tfidf_configs)
```

retriever = NearestNeighbors(retriever_configs)**

Se definen dos diccionarios: tfidf_configs que contiene la configuración para el vectorizador TF-IDF, y retriever_configs que contiene la configuración para el algoritmo de búsqueda de vecinos más cercanos. Luego, se instancian un objeto TfidfVectorizer y un objeto NearestNeighbors utilizando las respectivas configuraciones definidas.

Entrenar el modelo de recuperación de información:

X = embedding.fit_transform(documents['context'])

retriever.fit(X, documents['c_id'])

Se utiliza el método fit_transform del vectorizador TF-IDF para transformar los documentos en una representación numérica y se almacena en la variable X. Luego, se entrena el modelo de búsqueda de vecinos más cercanos utilizando el método fit de retriever con los datos transformados X y los identificadores de los documentos documents['c_id'].

Definir una función para transformar texto en vectores TF-IDF:

```
def transform_text(vectorizer, text):
    print('Text:', text)
    vector = vectorizer.transform([text])
    vector = vectorizer.inverse_transform(vector)
    print('Vect:', vector)
```

Esta función toma un vectorizador (vectorizer) y un texto como entrada. Imprime el texto y luego transforma el texto en un vector TF-IDF utilizando el método transform del vectorizador. Luego, invierte la transformación para obtener las palabras correspondientes utilizando inverse_transform y finalmente imprime el vector resultante.

Transformar y comparar la pregunta con los documentos:

transform_text(embedding, question)

X = embedding.transform([question])

c_id = retriever.kneighbors(X, return_distance=False)[0][0]

selected = documents.iloc[c_id]['context']

Se utiliza la función transform_text para transformar la pregunta en un vector TF-IDF y mostrarlo en pantalla. Luego, se utiliza el vectorizador para transformar la pregunta en una representación numérica X. Se obtiene el índice del documento más similar a la pregunta utilizando el método kneighbors del modelo de búsqueda de vecinos más cercanos (retriever.kneighbors). El resultado es un arreglo de índices, y se selecciona el primer índice [0][0] y se utiliza para obtener el contexto del documento seleccionado del DataFrame documents y se almacena en la variable selected.

Calcular y mostrar la precisión superior (top accuracy):

acc = top_accuracy(y_test, y_pred)

print('Accuracy:', f'{acc:.4f}')

print('Quantity:', int(acc*len(y_pred)), 'from', len(y_pred))

Se utiliza la función top_accuracy para calcular la precisión superior (top accuracy) comparando los valores verdaderos y_test con las predicciones y_pred. Luego se imprime en pantalla la precisión superior, la cantidad de predicciones correctas y el total de predicciones.