PROJECT REPORT
Text-to-SQL Conversion Using Pre-trained Model

Suyash Kumar
IIIT Delhi

The Python code leverages a pre-trained model hosted on Hugging Face (KN123/nl2sql) to convert natural language text into SQL queries. This facilitates intuitive interaction with databases by allowing users to input queries in everyday language, which the model then translates into executable SQL code.

Model Details

- Model Source: Hugging Face Transformers
- Model ID: `KN123/nl2sql`
- Type: Seq2Seq (Sequence-to-Sequence)
- Purpose: The model is designed to convert natural language questions or commands into SQL queries, enabling users to interact with databases without specific SQL knowledge.

Pipeline and Direct Model Loading

The code utilizes the pipeline function from the transformers library for a straightforward application of the model, which simplifies the process of generating SQL queries from text. Additionally, the code demonstrates direct loading of model and tokenizer, providing more control over the encoding and decoding processes, which is crucial for custom implementations or when fine-tuning is necessary.

Database Integration

The script interacts with an SQLite database, demonstrating the creation of a table and insertion of data, which showcases a practical application of the model in handling dynamic database content. The user's script automatically populates the database with randomly generated sales data, enabling practical demonstrations of the model's capabilities in a controlled environment.

Alternative Models

Several models are available for similar tasks, each with unique features and capabilities:

1. T5 (Text-To-Text Transfer Transformer): It can be fine-tuned for SQL generation, offering flexibility and robust performance across various text tasks.
2. BART (Bidirectional and Auto-Regressive Transformers): Suitable for text generation tasks including SQL query generation.

Choice of Model

The chosen model, `KN123/nl2sql`, is selected for its direct applicability to SQL generation and ease of integration via the Hugging Face library. It abstracts much of the complexity involved in model training and inference, providing a straightforward interface for text-to-SQL tasks.

Use Case

A text-to-SQL query generator can be a powerful tool in a company, significantly enhancing productivity for employees across various departments. This technology allows non-technical staff, such as business analysts, project managers, or even HR personnel, to retrieve, analyze, and interpret data from databases without needing in-depth knowledge of SQL. By simply typing a natural language query, employees can generate precise SQL statements to access the required information. This not only reduces the dependency on specialized database administrators but also speeds up decision-making processes, as data can be accessed and analyzed in real-time. Additionally, it minimizes errors associated with manual query writing and enables quicker iterations on data-driven projects, ultimately leading to more efficient operations and better business outcomes.

Model Limitations and Fine Tuning

The KN123/nl2sql model, as a generic pre-trained model, might not perform optimally with specific schema or vocabulary that hasn't been encountered during its training. The code acknowledges this limitation and suggests fine-tuning the model with specific dataset data to improve accuracy. Fine-tuning would involve training the model further on a curated dataset that closely resembles the actual use case scenarios and database schema involved.

Conclusion

The integration of natural language processing models like KN123/nl2sql to convert text to SQL queries represents an advancement in making data querying more accessible and efficient. While the model provides a robust starting point, fine-tuning it on specific data remains crucial for achieving high accuracy and relevance in professional settings. This can greatly enhance productivity and accessibility in data-driven industries, democratising data query capabilities across various user levels.

Python Script

```
#for installing transformers module in the system
!pip install transformers torch

# Model: https://huggingface.co/KN123/nl2sql/tree/main

# Use a pipeline as a high-level helper
from transformers import pipeline

pipe = pipeline("text2text-generation", model="KN123/nl2sql")

# Load model directly
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM

tokenizer = AutoTokenizer.from_pretrained("KN123/nl2sql")
model = AutoModelForSeq2SeqLM.from_pretrained("KN123/nl2sql")

import sqlite3

# Connect to SQLite database (or create it if it doesn't exist)
conn = sqlite3.connect('sales.db')
cursor = conn.cursor()

# Create a table named 'sales'
cursor.execute('''
CREATE TABLE IF NOT EXISTS sales (
    product_id INTEGER PRIMARY KEY,
    color TEXT,
    size TEXT,
    category TEXT,
    item TEXT,
    price REAL,
    sales INTEGER
)
''')
conn.commit()

import random
```

```
# List of sample data to insert into the database
data = [
    ("Red", "M", "Men", "Tshirt", 19.99, 120),
    ("Blue", "L", "Women", "Pant", 29.99, 85),
    ("Green", "S", "Kid", "Shorts", 15.99, 60),
    ("Yellow", "XL", "Men", "Shoes", 39.99, 50),
]


while len(data) < 20:
    color = random.choice(["Red", "Blue", "Green", "Yellow",
"Black", "White"])
    size = random.choice(["S", "M", "L", "XL"])
    category = random.choice(["Men", "Women", "Kid"])
    item = random.choice(["Tshirt", "Pant", "Shorts", "Shoes"])
    price = round(random.uniform(10, 50), 2)
    sales = random.randint(20, 150)
    data.append((color, size, category, item, price, sales))

# Insert data into the sales table
cursor.executemany('''
INSERT INTO sales (color, size, category, item, price, sales)
VALUES (?, ?, ?, ?, ?, ?)
''', data)
conn.commit()

def text_to_sql(text):
    # Prepare the input text and encode it to tensor
    inputs = tokenizer(text, return_tensors="pt",
max_length=512, truncation=True)

    # Generate SQL query using the model
    outputs = model.generate(**inputs)

    # Decode the generated ids to a SQL query
    sql_query = tokenizer.decode(outputs[0],
skip_special_tokens=True)
    return sql_query

# Test with an example
```

```
# example_text = "Show me all rows from sales where the category
is Women"

example_text = input()
sql_query = text_to_sql(example_text)
print("Generated SQL Query directly from model:", sql_query)

def execute_query(query):
    cursor.execute(query)
    # Fetch all results
    rows = cursor.fetchall()
    for row in rows:
        print(row)

execute_query(sql_query)
```