Customer Queries Text Classification

With Transformer and Fine-Tuning-DistilBERT

Bob Xue

Overview

Customer Queries come from Customer service support platforms, Text classification can be used to automatically categorize customer support queries, prioritize tickets and provide more timely and relevant responses.

Text classification is the process of assigning appropriate category from a pre-defined set of categories or tags to unstructured text data. It's commonly used as a supervised learning technique, which means that the algorithm is trained on a set of texts that have already been labeled with their respective categories. Once it's been trained on this data, the algorithm can use what it's learned to make predictions about new, unlabeled texts.

BERT, short for Bidirectional Encoder Representations from Transformers, is a powerful natural language processing (NLP) model developed by Google that uses a deep neural network architecture to understand the context of words in a sentence by looking at the surrounding words in both directions.

One of the most interesting things about BERT is that it's a pre-trained model. This means that BERT can be trained on massive amounts of text data, such as books, articles, and websites, before it's fine-tuned for specific downstream NLP tasks, including text classification. Once pre-trained, BERT can be fine-tuned for specific tasks, which allows it to adapt to the specific nuances of the task and improve its accuracy.

DistilBERT is Smaller, faster version of BERT, Comparable to BERT with 40% fewer

Parameters, commonly use in Resource constrained environments, mobile applications.

The objective of the project is to classify Customer Queries into pre-defined categories, so carried out the following steps in project notebook:

- Step 01: Import all the necessary libraries
- Step 02: Data Preparation
- Step 03: Create Features
- Step 04: Tokenization
- Step 05: Load the Model
- Step 06: Training
- Step 07: Evaluation
- Step 08: Inference

Detailed the above steps

Step 01: Import all the necessary libraries

```
import os
import warnings
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

from datasets import Dataset, DatasetDict
from datasets.features import Value, ClassLabel
from datasets import Features

from transformers import DistilBertTokenizerFast
from transformers import DistilBertTokenizer,
DistilBertForSequenceClassification

from transformers import TrainingArguments
from transformers import Trainer
```

```
from transformers import DataCollatorWithPadding
import torch
```

This step also set up the CUDA environment parameters:

```
os.environ['CUDA_LAUNCH_BLOCKING'] = '1'
os.environ['TF_FORCE_GPU_ALLOW_GROWTH']='true'
os.environ["PYTORCH_CUDA_ALLOC_CONF"] = "max_split_size_mb:512"
```

Step 02: Data Preparation

The Data Original source:
https://huggingface.co/datasets/coeuslearning/customerqueries

The dataset of choice determines the set of categories:
{0: 'Printer',
1: 'Scanner',
2: 'Laptop',
3: 'Monitor',
4: 'Keyboard',
5: 'Mouse',
6: 'Projector',
7: 'Fax machine',
8: 'Calculator',
9: 'Shredder',
10: 'Photocopier',
11: 'Whiteboard',
12: 'Paper shredder',
13: 'Desk lamp',
14: 'External hard drive',
15: 'Conference phone',
16: 'Label maker',
17: 'Document camera',
18: 'Wireless presenter',
19: 'USB hub'}

Step 03: Create Features

There are two features, first is "Query" which is type string and the other is the target "label".

```
features=Features({"Query": Value(dtype='string', id=None),
                "label": ClassLabel(num_classes=20,

names=['Printer','Scanner','Laptop','Monitor','Keyboard','Mouse','Projector',
'Fax machine','Calculator','Shredder','Photocopier','Whiteboard','Paper
shredder','Desk lamp','External hard drive','Conference phone','Label
maker','Document camera','Wireless presenter','USB hub'], id=None)})
```

This step also use pandas to directly convert from pandas's dataframe to dataset*:*

```
train_dataset = Dataset.from_pandas(train_df,
features=features,preserve_index=False)

test_dataset = Dataset.from_pandas(test_df,
features=features,preserve_index=False)

eval_dataset = Dataset.from_pandas(eval_df,
features=features,preserve_index=False)
```

Step 04: Tokenization

Use the BERT tokenizer to convert text into tokens that BERT can understand. This includes adding special tokens, padding, and truncating to a fixed length. Use DistilBert, a derivative of BERT model.

```
def tokenize_function(examples):

    return tokenizer(examples["Query"], truncation=True,
padding="max_length", max_length=256)


tokenizer = DistilBertTokenizerFast.from_pretrained("distilbert-base-
uncased")

# tokenized_datasets = dataset.map(tokenize_function, batched=True,
batch_size=1000)

tokenized_datasets = dataset.map(tokenize_function, batched=True,
batch_size=100)
```

```
# Dynamic Padding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

Step 05: Load the Model

Initialize the model with num_labels=20 as we have 20 classes to predict:

```
model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased", num_labels=20)

model.to(device)  # for Cuda
```

Step 06: Training

First, Define TrainingArugments:

```
training_args = TrainingArguments(output_dir="./results",
                                  per_device_train_batch_size=32,
                                  per_device_eval_batch_size=16,
                                  learning_rate=5e-5,
                                  num_train_epochs=5,
                                  optim='adamw_torch_fused',
                                  logging_dir="./logs/",
                                  logging_strategy='steps',
                                  logging_steps=200,
                                  evaluation_strategy='epoch',
                                  save_strategy='epoch',
                                  save_total_limit=2,
                                  load_best_model_at_end=True)
```

Second, Initialize the Trainer class:

```
trainer = Trainer(model=model,
                  args=training_args,
                  train_dataset=tokenized_datasets['train'],
                  eval_dataset=tokenized_datasets['validation'],
                  data_collator=data_collator,
                  tokenizer=tokenizer)
```

Last, Do the training:

```
%%time

trainer.train()
```

It spent time is 7min 48s, before adjust the TrainingArugments, the time was 14min 32s.

Step 07: Evaluation

Evaluate the model on the test set:

```
eval_results = trainer.evaluate()

print(f"Evaluation results: {eval_results}")
```

```
100%|██████████| 100/100 [00:07<00:00, 13.15it/s]

Evaluation results: {'eval_loss': 0.00147521635517478, 'eval_runtime':
7.7168, 'eval_samples_per_second': 207.34, 'eval_steps_per_second': 12.959,
'epoch': 5.0}
```

Step 08: Inference

Use the trained model to make predictions on new data:

```
model.eval()

# Example usage
# Inference
query = "How do I connect my printer to Wi-Fi?"
inputs = tokenizer(query, return_tensors="pt")

# Move input tensors to the same device as the model
inputs = {k: v.to(device) for k, v in inputs.items()}

with torch.no_grad():
    outputs = model(**inputs)

prediction = torch.argmax(outputs.logits).item()

print(f'Predicted Product: {prediction}')
```

```
Predicted Product: 0
```

After project improve thoughts

RAG consists of two components: a retrieval model and an answer generation model based on LLM. The retrieval model fetches context relevant to the user's query.

- **Reinforcement Learning for Optimizing RAG for Domain Chatbots**

Acknowledgements

- **Ecommerce Text Classification dataset**
- **Machine Learning in eCommerce Product Categorization by Nikolay Sidelnikov**

References

- **Text Classification with BERT**
- **DistilBERT**
- **Fine-Tuning-DistilBERT-for-Text-Classification**
- **E-Commerce Text Classification using Transformers**
- **Hyperparameter optimization**
- **Product classification**