text_classification

October 9, 2023

[ ]:
```
# Copyright 2023 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
```

1 Text Classification with Generative Models on Vertex AI

Run in Colab

View on GitHub

Open in Vertex AI Workbench

1.1 Overview

Generative models like PaLM 2 are powerful language models used for various natural language processing (NLP) tasks. One of those is text classification, which involves assigning one or more categories to a given piece of text. Although text classification can be done using traditional NLP techniques, LLMs can perform classification by providing prompts (as opposed to domain-specific labeled data), which can accelerate the time it takes to build a text classification solution. Classification models based on LLMs can be further tuned with many examples via custom model training, but that is beyond the scope of this notebook.

In this notebook, you will explore how to do text classification using prompts with the PaLM API. Learn more about classification prompts in the official documentation.

1.1.1 Objective

By the end of the notebook, you should be able to use a large language model to perform various classification tasks, including:

- Zero-shot prompting text classification
- Few-shot prompting text classification
- Common tasks:
  - Sentiment analysis
  - Topic classification
  - Spam detection
  - Intent recognition
  - Language identification
  - Toxicity detection
  - Emotion detection

1.2 Getting Started

1.2.1 Install Vertex AI SDK

```
[ ]: !pip install google-cloud-aiplatform --upgrade --user
```

Colab only: Uncomment the following cell to restart the kernel or use the button to restart the kernel. For Vertex AI Workbench you can restart the terminal using the button on top.

```
[ ]: # # Automatically restart kernel after installs so that your environment can␣
      ↪access the new packages
     # import IPython

     # app = IPython.Application.instance()
     # app.kernel.do_shutdown(True)
```

1.2.2 Authenticating your notebook environment

- If you are using **Colab** to run this notebook, uncomment the cell below and continue.
- If you are using **Vertex AI Workbench**, check out the setup instructions here.

```
[ ]: # from google.colab import auth
     # auth.authenticate_user()
```

1.2.3 Import libraries

Colab only: Uncomment the following cell to initialize the Vertex AI SDK. For Vertex AI Workbench, you don't need to run this.

```
[ ]: # import vertexai

     # PROJECT_ID = "[your-project-id]"  # @param {type:"string"}
     # vertexai.init(project=PROJECT_ID, location="us-central1")
```

```
[ ]: import pandas as pd
     from vertexai.language_models import TextGenerationModel
```

1.2.4 Import models

```
[ ]: generation_model = TextGenerationModel.from_pretrained("text-bison@001")
```

1.3 Text Classification

In the section below, you will explore zero-shot prompting, few-shot prompting, and some common types of text classification tasks.

1.3.1 Zero-shot prompting

Zero-shot prompting is where you do not provide examples with labels, and rely on the LLM to make the classification on its own.

```
[ ]: prompt = """
Classify the following:\n
text: "I saw a furry animal in the park today with a long tail and big eyes."
label: dogs, cats
"""

print(
    generation_model.predict(
        prompt=prompt,
        max_output_tokens=256,
        temperature=0.1,
    ).text
)
```

1.3.2 Few-shot prompting

With few-shot prompting, you provide examples to the PaLM model and expect it to predict classes based on the provided examples.

```
[ ]: prompt = """
What is the topic for a given news headline? \n
- business \n
- entertainment \n
- health \n
- sports \n
- technology \n\n

Text: Pixel 7 Pro Expert Hands On Review. \n
The answer is: technology \n

Text: Quit smoking? \n
The answer is: health \n

Text: Birdies or bogeys? Top 5 tips to hit under par \n
The answer is: sports \n
```

```
Text: Relief from local minimum-wage hike looking more remote \n
The answer is: business \n

Text: You won't guess who just arrived in Bari, Italy for the movie premiere. \n
The answer is:
"""

print(
    generation_model.predict(
        prompt=prompt,
        max_output_tokens=256,
        temperature=0.1,
    ).text
)
```

1.3.3 Other classification examples

Explore some more common text classification prompts below, which are all based on zero-shot prompts. You can also turn some of these into few-shot prompts by providing your own custom examples of text and the associated output classes.

Topic classification

[ ]:
```
prompt = """
Classify a piece of text into one of several predefined topics, such as sports,␣
↪politics, or entertainment. \n
text: President Biden will be visiting India in the month of March to discuss a␣
↪few opportunites. \n
class:
"""

print(
    generation_model.predict(
        prompt=prompt,
        max_output_tokens=256,
        temperature=0.1,
    ).text
)
```

Spam detection

[ ]:
```
prompt = """
Given an email, classify it as spam or not spam. \n
email: hi user, \n
      you have been selected as a winner of the lotery and can win upto 1␣
↪million dollar. \n
      kindly share your bank details and we can proceed from there. \n\n
      from, \n
      US Official Lottry Depatmint
"""

print(
    generation_model.predict(
        prompt=prompt,
        max_output_tokens=256,
        temperature=0.1,
    ).text
)
```

Intent recognition

```
[ ]: prompt = """
     Given a user's input, classify their intent, such as "finding information",␣
      ↪"making a reservation", or "placing an order". \n
     user input: Hi, can you please book a table for two at Juan for May 1?
     """

     print(
         generation_model.predict(
             prompt=prompt,
             max_output_tokens=256,
             temperature=0.1,
         ).text
     )
```

Language identification

```
[ ]: prompt = """
     Given a piece of text, classify the language it is written in. \n
     text: Selam nasıl gidiyor?
     language:
     """

     print(
         generation_model.predict(
             prompt=prompt,
             max_output_tokens=256,
             temperature=0.1,
         ).text
     )
```

Toxicity detection

```
[ ]: prompt = """
     Given a piece of text, classify it as toxic or non-toxic. \n
     text: i love sunny days
```

```
"""

print(
    generation_model.predict(
        prompt=prompt,
        max_output_tokens=256,
        temperature=0.1,
    ).text
)
```

Emotion detection

```
[ ]: prompt = """
Given a piece of text, classify the emotion it conveys, such as happiness, or␣
  ↪anger. \n
text: I'm still so delighted from yesterday's news
"""

print(
    generation_model.predict(
        prompt=prompt,
        max_output_tokens=256,
        temperature=0.1,
    ).text
)
```

1.3.4 Evaluation

You can evaluate the outputs of the text classification task if the ground truth classes are available. To showcase how this works, start by creating a simple dataframe with product reviews and the ground truth sentiment.

```
[ ]: review_data = {
    "review": [
        "i love this product. it does have everything i am looking for!",
        "all i can say is that you will be happy after buying this product",
        "its way too expensive and not worth the price",
        "i am feeling okay. its neither good nor too bad.",
    ],
    "sentiment_groundtruth": ["positive", "positive", "negative", "neutral"],
}

review_data_df = pd.DataFrame(review_data)
review_data_df
```

Now that you have the data with reviews and sentiments as ground truth labels, you can call the text generation model to each review row using the `apply` function. Each row will use the prompt in the `review` column to predict the sentiment using the PaLM API, and store the results in `sentiment_prediction` column.

```
[ ]: def get_sentiment(row):
         prompt = f"""Classify the sentiment of the following review as "positive",␣
      ↪"neutral" and "negative". \n\n
                     review: {row} \n
                     sentiment:
                   """
         response = generation_model.predict(prompt=prompt).text
         return response


     review_data_df["sentiment_prediction"] = review_data_df["review"].
      ↪apply(get_sentiment)
     review_data_df
```

In the end, you can call the `classification_report` function from sklearn to measure the accuracy and other classification metrics by passing ground truth sentiments `sentiment_groundtruth` and predicted sentiment `sentiment_prediction`:

```
[ ]: from sklearn.metrics import classification_report

     print(
         classification_report(
             review_data_df["sentiment_groundtruth"],␣
      ↪review_data_df["sentiment_prediction"]
         )
     )
```