#ml/nlp/openai

Examples

location:

- `~/scottdev/openai-cookbook/examples`

Classification using the embeddings

In the classification task we predict one of the predefined categories given an input. We will predict the score based on the embedding of the review's text, where the algorithm is correct only if it guesses the exact number of stars. We split the dataset into a training and a testing set for all the following tasks, so we can realistically evaluate performance on unseen data. The dataset is created in the Obtain_dataset Notebook.

In the following example we're predicting the number of stars in a review, from 1 to 5.

Clustering for Transaction Classification

This notebook covers use cases where your data is unlabelled but has features that can be used to cluster them into meaningful categories. The challenge with clustering is making the features that make those clusters stand out human-readable, and that is where we'll look to use GPT-3 to generate meaningful cluster descriptions for us. We can then use these to apply labels to a previously unlabelled dataset.

To feed the model we use embeddings created using the approach displayed in the notebook Multiclass classification for transactions Notebook, applied to the full 359 transactions in the dataset to give us a bigger pool for learning

Clustering

We use a simple k-means algorithm to demonstrate how clustering can be done. Clustering can help discover valuable, hidden groupings within the data. The dataset is created in the Obtain_dataset Notebook.

Code search

We index our own openai-python code repository, and show how it can be searched. We implement a simple version of file parsing and extracting of functions from python files.

Customizing embeddings

This notebook demonstrates one way to customize OpenAI embeddings to a particular task.

The input is training data in the form of [text_1, text_2, label] where label is +1 if the pairs are similar and -1 if the pairs are dissimilar.

The output is a matrix that you can use to multiply your embeddings. The product of this multiplication is a 'custom embedding' that will better emphasize aspects of the text relevant to your use case. In binary classification use cases, we've seen error rates drop by as much as 50%.

In the following example, I use 1,000 sentence pairs picked from the SNLI corpus. Each pair of sentences are logically entailed (i.e., one implies the other). These pairs are our positives (label = 1). We generate synthetic negatives by combining sentences from different pairs, which are presumed to not be logically entailed (label = -1).

For a clustering use case, you can generate positives by creating pairs from texts in the same clusters and generate negatives by creating pairs from sentences in different clusters.

With other data sets, we have seen decent improvement with as little as ~100 training examples. Of course, performance will be better with more examples.

Fine tuning classification example

We will fine-tune an ada classifier to distinguish between the two sports: Baseball and Hockey.

Get embeddings

The function `get_embedding` will give us an embedding for an input text.

How to handle rate limits

When you call the OpenAI API repeatedly, you may encounter error messages that say `429: 'Too Many Requests'` or `RateLimitError`. These error messages come from exceeding the API's rate limits.

Rate limits are a common practice for APIs, and they're put in place for a few different reasons.

- First, they help protect against abuse or misuse of the API. For example, a malicious actor could flood the API with requests in an attempt to overload it or cause disruptions in service. By setting rate limits, OpenAI can prevent this kind of activity.
- Second, rate limits help ensure that everyone has fair access to the API. If one person or organization makes an excessive number of requests, it could bog down the API for everyone else. By throttling the number of requests that a single user can make, OpenAI ensures that everyone has an opportunity to use the API without experiencing slowdowns.
- Lastly, rate limits can help OpenAI manage the aggregate load on its infrastructure. If requests to the API increase dramatically, it could tax the servers and cause performance issues. By setting rate limits, OpenAI can help maintain a smooth and consistent experience for all users.

Although hitting rate limits can be frustrating, rate limits exist to protect the reliable operation of the API for its users.

In this guide, we'll share some tips for avoiding and handling rate limit errors.

How to stream completions

By default, when you send a prompt to the OpenAI Completions endpoint, it computes the entire completion and sends it back in a single response.

If you're generating very long completions from a davinci-level model, waiting for the response can take many seconds. As of Aug 2022, responses from `text-davinci-002` typically take something like ~1

second plus ~2 seconds per 100 completion tokens.

If you want to get the response faster, you can 'stream' the completion as it's being generated. This allows you to start printing or otherwise processing the beginning of the completion before the entire completion is finished.

To stream completions, set `stream=True` when calling the Completions endpoint. This will return an object that streams back text as data-only server-sent events.

Note that using `stream=True` in a production application makes it more difficult to moderate the content of the completions, which has implications for approved usage.

Below is a Python code example of how to receive streaming completions.

Multiclass Classification for Transactions

For this notebook we will be looking to classify a public dataset of transactions into a number of categories that we have predefined. These approaches should be replicable to any multiclass classificaiton use case where we are trying to fit transactional data into predefined categories, and by the end of running through this you should have a few approaches for dealing with both labelled and unlabelled datasets.

The different approaches we'll be taking in this notebook are:

- **Zero-shot Classification:** First we'll do zero shot classification to put transactions in one of five named buckets using only a prompt for guidance
- **Classification with Embeddings:** Following this we'll create embeddings on a labelled dataset, and then use a traditional classification model to test their effectiveness at identifying our categories
- **Fine-tuned Classification:** Lastly we'll produce a fine-tuned model trained on our labelled dataset to see how this compares to the zero-shot and few-shot classification approaches

Question Answering using Embeddings

Many use cases require GPT-3 to respond to user questions with insightful answers. For example, a customer support chatbot may need to provide answers to common questions. The GPT models have picked up a lot of general knowledge in training, but we often need to ingest and use a large library of more specific information.

In this notebook we will demonstrate a method for enabling GPT-3 able to answer questions using a library of text as a reference, by using document embeddings and retrieval. We'll be using a dataset of Wikipedia articles about the 2020 Summer Olympic Games. Please see this notebook to follow the data gathering process.

Recommendation using embeddings and nearest neighbor search

Recommendations are widespread across the web.

- 'Bought that item? Try these similar items.'
- 'Enjoy that book? Try these similar titles.'
- 'Not the help page you were looking for? Try these similar pages.'

This notebook demonstrates how to use embeddings to find similar items to recommend. In particular, we use AG's corpus of news articles as our dataset.

Our model will answer the question: given an article, what other articles are most similar to it?

Regression using the embeddings

Regression means predicting a number, rather than one of the categories. We will predict the score based on the embedding of the review's text. We split the dataset into a training and a testing set for all of the following tasks, so we can realistically evaluate performance on unseen data. The dataset is created in the Obtain_dataset Notebook.

We're predicting the score of the review, which is a number between 1 and 5 (1-star being negative and 5-star positive).

Semantic text search using embeddings

We can search through all our reviews semantically in a very efficient manner and at very low cost, by simply embedding our search query, and then finding the most similar reviews. The dataset is created in the Obtain_dataset Notebook.

[1]: