Software Review

Journal of Educational and Behavioral Statistics
Vol. 00, No. 0, pp. 1–23
DOI: 10.3102/10769986241279927
Article reuse guidelines: sagepub.com/journals-permissions

Automatic Text Classification With Large Language Models: A Review of `openai` for Zero- and Few-Shot Classification

Kylie L. Anglin
Claudia Ventura
University of Connecticut

While natural language documents, such as intervention transcripts and participant writing samples, can provide highly nuanced insights into educational and psychological constructs, researchers often find these materials difficult and expensive to analyze. Recent developments in machine learning, however, have allowed social scientists to harness the power of artificial intelligence for complex data categorization tasks. One approach, supervised learning, supports high-performance categorization yet still requires a large, hand-labeled training corpus, which can be costly. An alternative approach—zero- and few-shot classification with pretrained large language models—offers a cheaper, compelling alternative. This article considers the application of zero-shot and few-shot classification in educational research. We provide an overview of large language models, a step-by-step tutorial on using the Python openai package for zero-shot and few-shot classification, and a discussion of relevant research considerations for social scientists.

Keywords: *large language models; LLMs; artificial intelligence; `openai`; educational measurement*

Natural language texts have long been a rich, yet challenging, source of data for social scientists. While text documents can provide valuable insights into educational and psychological constructs—beyond traditional tests and surveys alone—they are often expensive to analyze. For example, although verbatim transcripts of therapeutic sessions can offer highly nuanced information about the implementation of a specific therapy program—in far greater detail than a simple, therapist-reported fidelity checklist—incorporating transcript data into quantitative analyses often requires expert review according to a rating system (e.g., a rubric; Can et al., 2016). Similarly, while textbooks contain detailed information on the curricula that students receive, conducting extensive content analyses of educational materials (such as the various history textbooks used in

Texas) may be prohibitively time consuming if performed by hand (Lucy et al., 2020).

Because of these challenges, researchers increasingly use a *supervised learning* approach to measure educational and psychological constructs within natural language data. This involves manually labeling a random subset of documents according to a particular construct of interest and then teaching a machine learning algorithm to identify the same construct in a remaining, unlabeled sample. Such an approach can dramatically increase the scalability of measures drawn from unstructured text data; once the model has been trained, it may be applied repeatedly at negligible cost. Nevertheless, using supervised learning for measurement in the social sciences presents significant feasibility challenges. Chief among these challenges is the need for large amounts of *training data*. Achieving sufficient model performance—model output that closely matches the gold standard human ratings—often requires a large amount of text, pre-labeled by experts, for the model to learn from. Further, given the limited generalizability of most supervised learning models, researchers commonly require a new, hand-labeled dataset for every additional measurement task. As a result, although the use of text classification for measurement is increasing among social scientists, it remains relatively rare (Mcfarland et al., 2021).

One potential solution to these challenges involves *zero-shot* and *few-shot classification* via large language models. In zero-shot classification, a large language model will classify texts according to a construct that it has not been specifically trained to identify. In few-shot classification, a model will classify texts using a few examples and non-examples of a construct. This is possible because large language models are pretrained on enormous samples of language data. ChatGPT, for example, is trained on more than a million online documents (Radford et al., 2018, 2019). Through this pretraining, the model learns generalizable features of a language, including complex representations of words (used to capture semantic meaning) and patterns for parsing grammatical dependencies (used to identify the relationship between words in a text). In the case of generative artificial intelligence, the model can therefore generate an appropriate response with high frequency, even in response to a prompt it has never seen before—as demonstrated in the chat capabilities of ChatGPT (Ouyang et al., 2022).

Applied to text classification, this capability means that a model can successfully classify texts into provided categories, even in the absence of context-specific training data. Indeed, models that have been pretrained on large, general-use corpora often outperform models that have been specifically trained for the task at hand (Brown et al., 2020). Large language models can thus dramatically reduce the amount of expert-labeled data required to apply text classification; researchers may only need enough examples to validate the model's

results, rather than the hundreds or thousands of examples that are required to train a model from scratch.

Several resources are now available for understanding how large language models work. These resources cover the foundations of natural language processing (Manning & Schütze, 1999), neural networks, and related algorithms (Deng & Liu, 2018; Goodfellow et al., 2016) and include academic papers on the recent technological developments powering ChatGPT (Brown et al., 2020; Radford et al., 2018, 2019; Vaswani et al., 2017). In this article, we introduce and review the `openai` Python package for accessing Generative Pretrained Transformer (GPT) models that support zero- and few-shot text classification. We begin with a concise overview of large language models. Next, we describe the components of an `openai` program for text classification and demonstrate the available options for zero- and few-shot classification via an illustrative example. Finally, we discuss some broader considerations in the use of `openai` and zero-shot and few-shot classification in the social sciences.

A Short Introduction to Large Language Models

Some of the best-known large language models—such as GPT and Bidirectional Encoder Representations from Transformers (BERT)—can be understood as *foundation models*. Such models have been trained on massive amounts of data and can thus be adapted to a variety of downstream tasks (Bommasani et al., 2021). While only a few private companies and highly funded organizations have the computing resources to train these models, their straightforward adaptability has led to widespread use. GPT models, for example, can be adapted to new downstream tasks simply by providing them with a natural language prompt (Bommasani et al., 2021; Radford et al., 2019).

Although large language foundation models share many similarities (such as their transformer-based model architectures), one important distinction is worth emphasizing. While some models, like GPT, are trained to be generative, others, like BERT, are trained to be discriminative. Generative models are designed to produce a ‘‘reasonable continuation’’ of natural language text (Reynolds & McDonell, 2021). Discriminative models, on the other hand, are designed to discriminate between classes (Bommasani et al., 2021). Thus, discriminative models like BERT are not as well-suited for tasks that directly involve text generation, such as translation or chat, which generative models like GPT handle quite effectively. However, BERT’s bidirectional architecture (which allows the algorithm to simultaneously consider text surrounding both sides of a given term) makes it a powerful tool for taking context into account in discrimination tasks (Devlin et al., 2018), one reason why it plays an important role in ranking within the Google Search engine (Nayak, 2019).

Given the distinction between generative and discriminative models, one might suppose that discriminative models are the most appropriate choice for text classification, as classification is a discriminative task. However, employing a generative approach to classification has certain advantages. First, compared to discriminative foundation models, it is often easier to adapt a generative foundation model to a new classification task. For example, the standard adaptation approach with a GPT model is to simply write a prompt describing the classification task, such as: ''Classify the following text into one of two categories …'' (Brown et al., 2020; Plaza-del-Arco et al., 2022). The standard adaptation approach with BERT, on the other hand, involves retraining the pretrained model for additional epochs using context-specific labeled data—a process known as fine-tuning (Mosbach et al., 2020). Compared to GPT's uncomplicated ''prompt engineering,'' such fine-tuning requires greater resources and technical skill. Another advantage of using generative models is that these models are demonstrably more stable than fine-tuned, discriminative models, particularly when the fine-tuning dataset is small (Mosbach et al., 2020; Yogatama et al., 2017). Third, and relatedly, generative models may be particularly advantageous in the case of imbalanced data. When examples of the construct of interest are rare, it takes even larger amounts of hand-labeled data to generate enough examples to train or fine-tune a discriminative model effectively (Fernández et al., 2018). A generative model, on the other hand, may only require a well-crafted prompt and/or a few choice examples of the rare construct (Brown et al., 2020).

Generative Large Language Models for Text Classification

Though a generative language model is trained to produce reasonable continuations of text, this capability may be effectively exploited for the purposes of classification. To understand how, we first consider the standard generative language modeling objective. A generative model aims to maximize the probability of new language given previous language. Formally, the standard objective is to maximize the following likelihood for a given document, $D = \{u_i, u_2, \ldots u_n\}$, represented by n tokens (where tokens may be approximately understood as individual words, but in practice also include word fragments and common phrases; Sennrich et al., 2015):

$$p(D) = \sum_{i}^{n} p(u_i | u_{i-k}, \ldots, u_{i-1}; \theta). \tag{1}$$

Here, k is the size of a context window preceding token i, and the conditional probability of a token is modeled using a neural network with parameters θ (Radford et al., 2018, 2019). Put simply, the model aims to estimate the probability of a particular token, conditional on the previous tokens in a document.

Importantly, the information required to produce appropriate continuations of text is also useful for classifying unlabeled texts into defined categories. In fact, text classification may be viewed as a specific type of text continuation. To illustrate, Equation 1 may be understood as estimating $p(output|input)$; the probability of a text continuation given previous text (Radford et al., 2019). A generative model is more likely to produce the desired result, however, if task instructions are treated separately from text input; formally: $p(output|input, task)$. Here, the model estimates the most probable output given the inferred task—a problem termed *unsupervised task learning* (Radford et al., 2019).

For example, in a French-to-English translation task, the desired text continuation would be the English translation of a French input, with the task specified as ''translate to English.'' In text classification, the task might be specified as ''Classify the provided document into one of the two provided categories …'' Here, the inferred task is text classification, the input is the text, and the desired output is the predicted classification. In either case, the model is continuing the text when given a prompt; the prompt is simply separated into two pieces of information: task and input. This separation of task and input is embedded in OpenAI's Python module via system and user messages, a process described in detail shortly.

The Pretraining Task

In order to successfully generate text, OpenAI's latest models were trained on millions of documents scraped from the Internet (the Common Crawl corpus), together with the entirety of Wikipedia and Stack Exchange, and thousands of books (Radford et al., 2018; Ray, 2023). Using this training corpus, the models have been optimized to complete a *masked language modeling task*, which involves predicting the most likely missing word from a selection of text. For example, a hypothetical sentence in the corpora, ''I'll need to pick up some groceries from the store,'' might be provided to the model with a masked term, such as: ''I'll need to pick up some groceries from the [MASK].'' The words preceding [MASK] are treated as inputs to the model, and the model's objective is to correctly predict the masked word based on the inputs. In this example, if the model predicts ''store'' with a high probability, it would be considered a successful prediction. Further, to place a high probability on the word ''store,'' the model must have learned patterns related to the semantic relationship between terms (e.g., store and groceries) and the syntactic relationship between terms (e.g., determiners and nouns). This same linguistic information may prove useful for many text classification tasks. Thus, the model learns generalizable information without human supervision (Radford et al., 2018).

Major Components of a Transformer

Transformer models have four key components that together enable GPT models to produce reasonable continuations of text, even in situations featuring a never-before-seen prompt: word embeddings, positional encodings, attention, and neural networks.

Word Embeddings. Transformer models do not treat each token as a unique scalar value but as a word embedding—a multidimensional vector that captures semantic meaning. Word embeddings map individual tokens to numerical vectors of a predetermined length (over 12,000 dimensions in the case of GPT-4; OpenAI, 2024b). Subsequently, the semantic meaning of the word is represented by the coordinates of the vector, and the vector is optimized so that words that appear in similar contexts will be near each other in vector-space (Mikolov et al., 2013). For example, the words ‘‘groceries’’ and ‘‘store’’ will likely have a higher vector similarity than ‘‘groceries’’ and ‘‘park.’’ Word embeddings thus elegantly handle both direct synonyms and word similarity. The word embeddings themselves are commonly trained using a masked language modeling task similar to the process described above (Devlin et al., 2018).

Positional Encodings. Beyond the meaning of words, generative language models require a mechanism for incorporating information about word order. With transformer models, this occurs via a positional encoding: a certain value, added to each token’s embedding, that provides information on the token’s location. For this process, Vaswani et al. (2017) propose using sine and cosine functions of the token’s position, creating a positional encoding of the same dimension as the word embedding. When the word embedding and positional encoding are summed, the *positional embedding* that results contains information on both token meaning and token order.

Attention. A key challenge in the representation of text data concerns the handling of long-term dependencies in language. Because the meaning of a word often depends on information provided earlier in a text, a successful language model must relate individual positional embeddings to one another, even when the associated terms are separated by many tokens in the original natural language input. Transformers achieve this through an attention mechanism: a weighted average of the positional embeddings, where the weights are determined by the positional embeddings’ similarity to each other. In this process, words that are similar in meaning, and close to one another in the text, will have a higher weight. This simple self-attention mechanism can be improved with a *multihead attention mechanism* that enables the model to give multiple, variable

weights to individual tokens. The model can thus attend to text sequences within the input at different levels of abstraction, as if viewing the same word from multiple perspectives, depending on which surrounding words have been assigned the greatest weight (Radford et al., 2018).

Neural Networks. The final reason that large language models have become so successful at producing reasonable continuations of text is that the underlying neural network allows the model to learn complex linguistic patterns. Neural networks can be understood as multilayered systems of nonlinear functions which are optimized to reduce error (Derry et al., 2023). There are four fundamental components of a neural network: first, a data input layer; second, hidden layers of functions (often called ''neurons,'' although similarities to brain activity are limited; Chollet, 2021); third, assigned weights, representing the connections between functions; and fourth, an output layer that provides the prediction (Derry et al., 2023; Jain et al., 1996). Within the hidden layers, each ''neuron'' evaluates a simple function based on the neurons from the previous layer, and previous neurons only ''fire'' (or propagate) if the activation function's signal is strong enough. In this way, the output of one layer becomes the input for the next. Although each underlying function may be simple, connections between functions allow the network to learn complex patterns, particularly as weights are iteratively adjusted to reduce error. Therefore, when the transformer's neural network is fed with the output of the multihead attention mechanism applied to positional embeddings, the transformer can learn complex patterns of word meaning and word order, while also handling long-term dependencies.

Fine-Tuning With Reinforcement Learning

Before releasing ChatGPT to the public, OpenAI incorporated one further technological feature: fine-tuning the model using human feedback (Ouyang et al., 2022). Here, researchers began with a set of prompts, paired with human-generated appropriate responses. These pairs were then used to update model parameters, encouraging responses that are similar to those generated by humans (Ouyang et al., 2022). Next, the researchers collected multiple model outputs in response to prompts. Human labelers ranked these outputs from best to worst, and these data were used to teach a *reward model* to identify human preferences. The final model is optimized against the reward model using reinforcement learning (Ouyang et al., 2022; Schulman et al., 2017). This process results in a model that has been shown to be less likely to produce inaccuracies and more likely to align with human preferences (Ouyang et al., 2022).

Additional Advancements in the GPT Models

Overall, large language models hold great promise for text classification. This is because estimating $p(output|input, task)$ is possible when a complex enough architecture—featuring an approach to handling long-term dependencies—is given a large enough corpus to learn from. With each new release of a GPT model, from GPT-1 to the most recent GPT-4, OpenAI has improved these attributes; GPT-2 has 1.5 billion parameters, GPT-3 has 175 billion, and GPT-4 has 100 trillion (Brown et al., 2020; Ray, 2023). Given that performance generally scales with parameters, these are significant improvements (Brown et al., 2020). Further, the context size handled by GPT has also increased. The GPT-2 model allows for a prompt of 1,024 characters, while GPT-3 allows 2,048, and GPT-4 allows 8,195 (OpenAI, 2024b)—increasing the model's usefulness for performing text classification with long documents. Finally, all of the GPT models are trained on an immense corpus of text, which is updated with new online data following each release.

OpenAI Software

OpenAI, the artificial intelligence research organization that produced ChatGPT, provides a Python interface for programmatically interacting with their models. The Python library, `openai`, enables users to efficiently send multiple requests in quick succession, and to modify their interactions as required. For clarity, moving forward, we capitalize OpenAI when referring to the organization and use lowercase characters when referring to the Python library `openai`. Besides the Python library, OpenAI also provides a JavaScript library and several community libraries built and maintained by external users (OpenAI, 2024c). In the following sections, we first describe the preliminary steps a researcher must take to use the `openai` library. Second, we present the core elements of an interaction with an OpenAI model, such as GPT-4, through the `openai` library. Third, we illustrate the library's core functions and key user options via an example—classifying transcribed utterances within the context of an educational intervention. Finally, we discuss key considerations in the use of `openai` for classification and point researchers to additional resources.

Access and Installation

Unlike the chat function provided at chat.openai.com, programmatic use of the OpenAI application programming interface (API) is not currently free. Thus, the first step is to create an account with OpenAI and obtain an API key, a many-character code which acts as a personal authentication mechanism. The key must be integrated into code, as in Code Snippet 1, to enable communication with the API. Note that OpenAI will only display the key a single time, so

users must store it safely and avoid sharing the key with unauthorized users. After obtaining an API key, set up is straightforward because the `openai` package can be installed using pip, the go-to package management system for Python.

Code Snippet 1

Installation and Client Setup

```
!pip install openai
from openai import OpenAI
OPENAI_API_KEY = "YOURKEYHERE"
client = OpenAI(api_key=OPENAI_API_KEY)
```

Core Elements

Assuming a prepared text dataset, using `openai` for text classification requires two core tasks: first, sending a chat request to an OpenAI model; and second, extracting a response. Following the convention used by OpenAI, from now on, we will refer to the selected model as the *assistant*. In other words, the assistant provides the response to the user request.

Code Snippet 2

Example Chat with One User Message

```
response = client.chat.completions.create(
    model="gpt-4-turbo",
    messages=[
        {"role": "user",
        "content": "Which planet is closest to the sun?"}]
)
```

Chat Request. Users send requests to an assistant using the `chat.completions.create()` method, where the subject of the method is the client defined in Code Snippet 1. The body of the request requires two arguments: `model` and `messages`. The model is simply the name of the OpenAI model, which you would like to call. For example, users calling GPT-4 may use "gpt-4-turbo" as the relevant value. The `messages` argument contains the chat to which the assistant will respond.

Messages themselves consist of two objects: a `role` (either "`system`," "`user`," or "`assistant`") and `content`. The simplest message will consist of a single user role, with text input (i.e., the prompt) provided in the `content`. Code Snippet 2 illustrates this format: the message contains one role (here, user)

and one text prompt in the content (‘‘Which planet is closest to the sun?’’). The assistant may then respond with output, such as ‘‘Mercury is the planet that is closest to the sun.’’ In this case, running the code contained in Code Snippet 2 is equivalent to submitting a single chat to the assistant at chat.openai.com.

However, users can improve their control over the assistant response via the system role. The message contained in the system role is intended to place constraints on an appropriate response. For example, in Code Snippet 3, we tell the assistant to respond in JSON format. While the user role is analogous to the *input* in $p(output|input, task)$, the system role is analogous to the *task*; the model infers the task based on information provided in the system role content. For text classification, the system message commonly contains instructions informing the model that it should provide responses within a given format (e.g., True or False).

Code Snippet 3

Example Chat with System Message

```
response = client.chat.completions.create(
  model="gpt-4-turbo",
  messages=[{"role": "system",
             "content": "Provide your output in JSON format" \
                        "with the key planet."},
             {"role": "user",
             "content": "Which planet is closest to the sun?"},
        ]
)
```

Users may also provide examples of the desired behavior via the user and assistant roles. Here, the user provides input in the user role, alongside a desired response example from the assistant in the assistant role, as illustrated in Code Snippet 4. The final message should still be the prompt to which the assistant should respond. Thus, when examples are provided, a typical format begins with (a) a system message, followed by (b) alternating user and assistant messages (the examples), and ending on (c) a final user message (the prompt). For example, Code Snippet 4 contains a system message prompting the model to provide a response in JSON format, an example user input, an example assistant output demonstrating the desired format, and a final user input to which the assistant will respond.

Code Snippet 4

Example Chat with Assistant Message

```
response = client.chat.completions.create(
  model="gpt-4-turbo",
  temperature = 0.0,
  messages=[{"role": "system",
             "content": "Provide your output in" \
                        "JSON format with the key planet."},
            {"role": "user",
             "content": "Which planet is closest to the sun?"},
            {"role": "assistant",
             "content": "{"planet": "Mercury"}"},
            {"role": "user",
             "content": "Which planet is farthest from the sun?"}
        ]
)
```

Finally, users may also ask for multiple independent responses using the optional `n` argument and can control the temperature of the model (the amount of randomness allowed in selecting the most probable response) using the `temperature` argument. Note that a temperature greater than zero often results in seemingly more creative, human-like text, but this is rarely a goal of text classification. Message roles and their application to text classification can be found in Table 1.

Chat Completion Object. Assistant responses, and associated metadata, are provided in `ChatCompletion` objects. Figure 1 provides an example of a chat completion in response to Code Snippet 4. The most important information is contained in the `choices` array. If `n`, above, was set to 1 (the default parameter, indicating that the assistant should provide one response), then the response message can be extracted using `choices[0].message.content` (i.e., the message content of the first response, because Python indexing starts at 0). Other useful information contained in the `ChatCompletion` object includes the explanation for the end of text generation (ideally ‘‘stop’’ indicating that there was no error), token usage (relevant for pricing), and a unique identifier.

Applied Example

In this section, we demonstrate the use of the `openai` package for zero- and few-shot classification within the context of an educational intervention, classifying utterances from transcripts of a teacher training program. The overarching goal is to use text classification to automatically monitor the fidelity of coaches

TABLE 1.
Roles and Their Application to Text Classification

Role	Explanation	Example for text generation task	Example for text classification task
System	Provides instructions to the assistant on the desired behavior.	‘‘You are a 5th grader learning about space.’’	‘‘Respond with either Positive, Negative, or Neutral to all requests.’’
User	Prompt to which the assistant should respond.	‘‘Write a poem about the planets.’’	‘‘Classify the sentiment of the following text as Positive, Negative, or Neutral: The movie was alright.’’
Assistant	Used to provide examples of desired assistant responses.	‘‘Mercury so small, Venus glows like a bright ball, Mars got in a brawl.’’	‘‘Neutral’’

Figure 1

Example Response Object

```
ChatCompletion(id='chatcmpl-9X9lBzG7Vvi26j76CC2Obj54faeLh',
    choices=[Choice(finish_reason='stop',
        index=0,
        logprobs=None,
        message=ChatCompletionMessage(
            content="Mercury is the planet closest to the sun.',
            role='assistant",
            function_call=None,
            tool_calls=None))],
    created=1717689369,
    model="gpt-4-turbo-0125",
    object="chat.completion",
    system_fingerprint=None,
    usage=CompletionUsage(
        completion_tokens=10,
        prompt_tokens=15,
        total_tokens=25))
```

FIGURE 1. *Example response object.*

in their conversations with teachers-in-training (Boguslav, 2024; Cohen et al., 2020). Within the coaching conversation, coaches are expected to (a) affirm an effective teaching practice (a strategy we term positive evaluation), (b) identify a target skill, and then (c) engage the teacher in a roleplay so that the teacher can practice the targeted skill. Here, we demonstrate how `openai` may be used to automatically identify instances of coaches providing positive affirmation of a teaching practice.

Data Preparation, Prompt Engineering, and Performance Criteria

We draw from a corpus of transcripts that has been hand-coded by undergraduate research assistants in order to highlight a variety of coaching strategies. Here, we focus on one of these strategies: *positive evaluation*. Human coders were asked to classify coaches' turns-of-talk according to the following definition of positive evaluation provided in the codebook: ''The coach communicates a positive judgment about a teacher's skill, specific elements of the teacher's practice, or provides a general affirmation of the teacher's instruction in a specific lesson or time-period.'' Agreement among human coders was 0.96 (Krippendorff's $\alpha = .82$).

Within a hand-labeled dataset of 868 turns-of-talk, 117 are examples of positive evaluation. We split this hand-labeled data into training, development, and testing datasets, such that the training dataset contains 10 random positive examples and 10 random negative examples, the development dataset contains 40 positive examples and 40 negative examples, and the testing dataset contains all remaining examples. From the training data, we draw potential examples for few-shot classification (where the assistant is provided with a few examples of desired behavior). The development dataset is used to tinker with the prompt (a process termed prompt engineering), and the testing data are used to assess the alignment between assistant responses and our gold-standard human coded labels. Given imbalance in the data (only 13% of turns-of-talk contain instances of positive evaluation), our splitting scheme ensures that there are enough positive examples in the training dataset to use for few-shot classification and that there are enough positive examples in the development dataset to accurately estimate recall (percent of positive instances identified by the model). We store the labeled data as a pandas DataFrame named featuring an `id` column, `text` column, and `gold_standard` column.

We validate the model's performance within the testing dataset by assessing the relationship between true positives (TP; the number of human-identified instances of positive evaluation that are also identified by the model), false negatives (FN; human-identified instances of positive evaluation that are not identified by the model), true negatives (TN; instances which the model and humans both agree are not positive evaluations), and false negatives (FP;

model-identified instances of positive evaluation that a human did not identify as such). Using these values, we estimate:

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{TP} + \text{FN} + \text{TP}}, \tag{2}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{3}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{4}$$

Zero-Shot Classification

We first demonstrate a simple example of zero-shot classification. Table 2 provides the system role content (stored in Code Snippet 5 as `system_content`) and user role content (stored as `user_content`). The system role content contains a few key elements. First, we use the system role to provide context to the assistant, informing the assistant that the user will provide an excerpt of a conversation between a professional development coach and a teacher. Then, we indicate the desired task: we want the assistant to classify the excerpt as either an instance of positive evaluation or not. Third, we provide a definition of positive evaluation. This definition is taken directly from the same codebook provided to the human coders. Fourth, we provide an explanation of the desired format for the response (here, True or False). Given this system message, the user message simply contains a direct turn-of-talk pulled from a transcript. Note that we set temperature to zero to reduce the influence of randomness in the response; we want the assistant to strictly select the most probable text continuation given the prompt.

Code Snippet 5

Zero-Shot Classification Example

```
response = client.chat.completions.create(
    model="gpt-4-turbo",
    messages=[
        {"role": "system", "content": system_content},
        {"role": "user", "content": user_content},
    ],
    temperature = 0.0,
)
```

Note. The value of `system_content` is as specified in Table 2. The `user_content` contains the text to be classified.

TABLE 2.
Zero-Shot Classification Message Components for Illustrative Example

Role	Content	Python variable
System	You will be provided with an excerpt of a conversation between a professional development coach and a teacher. The excerpt is from the coach. Your job is to classify the excerpt according to whether it contains an instance of positive evaluation where ‘‘positive evaluation’’ is defined as: the coach communicates a positive judgment about a teacher’s general skill as a teacher, specific elements of the teacher’s practice, or provides a general affirmation of the teacher’s instruction in a specific lesson or time-period. Respond with either True or False.	`system_content`
User	Yes, yes. It’s very stressful. A lot of first-year teaching students struggle with it. So, I appreciate you pushing through it. And I think you did a good job when addressing the students when behaviors went on and they were not paying attention to what they were supposed to do. But then like you were able to address it. And you called them out and said, ‘‘hey, we’re supposed to be doing this right now, we need to focus’’ So, that is something I want you to continue to do.	`user_content`

Note. The text provided in the user row of the table has been edited from the original verbatim transcription to increase readability in the table. However, in practice, we do not make any changes to the transcript excerpts provided to the model in the `user_content`.

Few-Shot Classification

Table 3 provides the messages used to perform few-shot classification. We provide the assistant with one positive example and one negative example. The positive example classification pair is stored in `user_content1` and `assistant_content1`. The negative example classification pair is stored in `user_content2` and `assistant_content2`. The initial system message and final user message remain the same as in Code Snippet 5. The few-shot classification is implemented in Code Snippet 6.

Code Snippet 6

Few-Shot Classification Example

```
response = client.chat.completions.create(
    model="gpt-4-turbo",
    messages=[
        {"role": "system", "content": system_content},
        {"role": "user", "content": user_content1},
        {"role": "assistant", "content": assistant_content1},
        {"role": "user", "content": user_content2},
        {"role": "assistant", "content": assistant_content2},
        {"role": "user", "content": user_content},
    ],
    temperature=0.0,
)
```

Scaling to Full Corpus and Data Extraction

Code Snippet 7 contains the code that we use to apply few-shot classification to the full corpus of documents. We loop through each row in the dataset, `df`, then send a text completion request, and finally store the response in a list. The list of assistant classifications then becomes a new column in the pandas DataFrame. Table 4 provides the estimated performance metrics within our testing sample (with estimated standard errors in parentheses). Accuracy in both the zero- and few-shot cases is quite high: 0.96 and 0.97, respectively. Recall for both models is approximately 0.9, but the few-shot model is more precise (0.82 compared to 0.71).

Code Snippet 7

Few-Shot Classification Applied to Full Sample

```
import pandas as pd

classifications = []
for text in df["text"]:
    response = client.chat.completions.create(
        model="gpt-4-turbo",
        messages=[
            {"role": "system", "content": system_content},
            {"role": "user", "content": user_content1},
            {"role": "assistant", "content": assistant_content1},
            {"role": "user", "content": user_content2},
            {"role": "assistant", "content": assistant_content2},
            {"role": "user", "content": text},
        ],
        temperature=0.00,
    )
    classification = response.choices[0].message.content
    classifications.append(classification)
df["classification"] = classifications
```

TABLE 3.
Few-Shot Classification Message Components for Illustrative Example

Role	Content	Python variable
User	So, hopefully you we can talk about a few things just to make you feel even better about the next time yeah. So, first I want to say what a great job maintaining enthusiasm. I know that it's hard to do when you are in the simulator they're trying to interact with the avatars so great job there. Also, I really appreciated the way that you were constantly having um, you were probing for evidence. So, every time they were making a claim you asked saying to the students: What makes you think that? What in the text made you draw those conclusions? Where can you point me toward? Those are all really good things to think about student understanding and to extend instead of thinking so continue to do that.	`user_content1`
Assistant	True	`assistant_content1`
User	Great. So, you want to be able to, when the student gets an answer incorrect, to point them back to the text, so that same question of text evidence. You can ask them- if you're comfortable- you can ask them to provide text evidence. In this case I think Ava already gave the text evidence.	`user_content2`
Assistant	False	`assistant_content2`

Note. System role message and final user role content remains the same as in Table 2.

TABLE 4.
Model Performance

Model Type	Accuracy	Precision	Recall
Zero-shot	0.96 (0.01)	0.71 (0.05)	0.9 (0.04)
Few-shot	0.97 (0.01)	0.82 (0.04)	0.9 (0.04)

Note. Zero-shot performance is based on the prompt provided in Table 2. Few-shot performance is based on the prompt and examples provided in Table 3. Standard errors are in parentheses and are estimated using, $\sqrt{\frac{p(1-p)}{n}}$, where p is the performance metric and n is the number of observations in the denominator of Formulas 2 through 4.

Potential Extensions and Options for Improvement

Performance in our illustrative example is quite high but, ultimately, performance will vary depending on the complexity of the text classification task. Here, positive evaluation may be a relatively straightforward construct to identify in our dataset. When performance is found to be insufficient, researchers have a few key options. First, they may continue tinkering with the prompt while testing the performance of the model using the training or development dataset (still retaining the testing data for validating *final* model performance). Second, they may test several combinations of examples within the few-shot models, again assessing performance in the development dataset. Given few-shot learning involves only a few examples, the contents of those examples are particularly important for determining model performance as extraneous information can reduce model accuracy (Reynolds & McDonell, 2021).

Finally, if performance is still sub-standard, researchers also have the option of fine-tuning GPT models. OpenAI suggests saving fine-tuning as a last potential option, after iterating over prompts and/or examples because: (a) Fine-tuning may not be necessary given successful prompt engineering; (b) fine-tuning has a slower feedback loop compared to prompt engineering; and (c) even if fine-tuning is ultimately necessary, efforts spent in prompt engineering are not wasted as fine-tuning is best performed on an already strong prompt (OpenAI, 2024a). On the other hand, fine-tuning does offer a few advantages. Because OpenAI charges according to the length of messages, fine-tuning can reduce costs compared to few-shot learning with many examples. Fine-tuned models may also have lower latency, reducing the time it takes to apply the model to the full corpus. Finally, fine-tuning with a larger number of examples may ultimately improve model performance, particularly when the classification task is hard to articulate in the prompt (OpenAI, 2024a).

Discussion

Like any other classification model, the validity and performance of text classification with a GPT model can benefit from some form of a training/development/testing split. Of course, training in the few-shot case requires far less labeled data than traditional supervised learning; rather than collecting thousands of examples to train a model from scratch, the researcher needs only to identify a few key examples. Further, instead of tinkering with algorithms and hyperparameters, the researcher can use a development corpus to tinker with the prompt and the examples provided to the assistant—choosing the combination that results in the highest performance within the development data.

Regardless, the final step, validation, is as important as ever. Researchers need to know how well the model performs at identifying the construct of interest. We illustrated one validation mechanism here, comparing zero- and few-

shot classifications to those produced by human labelers. In this case, the few-shot classifier identified 90% of human-identified instances of positive evaluation, and, in 82% of cases where the model positively classified a turn of talk, human coders agreed that the turn of talk was positive evaluation.

The extent to which these metrics should be interpreted as evidence of model validity depends on at least three criteria. The first concerns whether the testing data were truly unseen during model development. If prompt engineering was conducted based on observations in the testing dataset, performance metrics may be overestimated. For this reason, we strongly suggest that researchers continue to split their data into training, development, and testing datasets, as is common practice when locally training supervised machine learning models (Hastie et al., 2009). Second, the estimates' precision is also relevant. The standard errors in Table 4 illustrate that performance metrics are point estimates, derived from a sample (the testing data). Like all point estimates, performance metrics are associated with uncertainty (Savoy, 1997). In our case, for example, the 95% confidence interval suggests that the true recall of the few-shot classifier lies somewhere between 0.82 and 0.98. Third, the validity of the human-labeled data is important to consider. Performance metrics are only indicators of validity if we believe that the human-labeled data are themselves trustworthy. To this end, we grounded our definition of positive evaluation in the literature (Boguslav, 2024), required the certification of coders, and assessed inter rater reliability (Anglin et al., 2022).

Beyond assessing model performance, researchers may also compare the performance of the classifier among key population subgroups. An ongoing concern with any application of artificial intelligence is the potential for biases in the trained data to be reflected, or amplified, in the model output (Baker & Hawn, 2021). In the case of text classification, this can result in differential classifier performance among demographic or linguistic subgroups. OpenAI has taken several measures to limit the biases of GPT models, including using reinforcement learning to penalize harmful responses (Ouyang et al., 2022). However, it is also the responsibility of researchers using GPT models to ensure that doing so does not create unintentional harm. Demonstrating sufficient performance among protected subgroups is one method of protection (Mitchell et al., 2019).

There are at least two additional considerations accompanying the use of `openai` for text classification: participant privacy and cost. With respect to participant privacy, language data provided to the assistant through the API will be temporarily shared with OpenAI. However, unlike input provided at chat.openai.com, input provided through the API is not currently retained by OpenAI. Indeed, OpenAI states that they ''securely retain API inputs and outputs for up to 30 days to provide the services and to identify abuse. After 30 days, API inputs and outputs are removed from our systems, unless [OpenAI] are legally required to retain them'' (OpenAI, 2023). Nonetheless, ongoing

consideration of privacy and data ownership is critical. At a minimum, researchers should consider removing any identifiable information from documents before using `openai`.

Finally, the costs of using the `openai` platform are currently negligible compared to the costs of human coding. Here, we classified 868 short documents (averaging approximately 45 words each) for $26.84, including the costs associated with prompt engineering. Classifying the same number of documents took our human coders approximately 10 hours, not including time spent hiring, training, and certifying the coders. To the extent that the GPT models can approximate the work of undergraduate coders, these cost savings are substantial. However, given the increasing importance of ensuring accuracy and unbiasedness in the results of large language models, we suggest that some of these retained resources should be diverted to support additional validation.

Documentation

We hope that this software review helps readers to gain a greater understanding of how large language models work, how their capabilities may be utilized for text classification, and the core operations for doing so within the `openai` Python package. However, given that OpenAI provides frequent updates to its models and best practices, we refer readers to the relevant `openai` documentation: https://platform.openai.com/docs/overview. There, readers can find information on the latest models and updates, as well as additional details for tailoring model prompts and responses.

Conclusion

In a relatively short period, large language models have revolutionized the field of natural language processing. The full implications of large language models and the increasing ability of artificial intelligence are far beyond the scope of this review. Here, however, we explore one especially useful application for behavioral and educational researchers: zero- and few-shot text classification. The `openai` Python package provides a straightforward API, which can be used to classify documents at high levels of performance, even with very limited training data. With corresponding validation, therefore, these models can provide new opportunities for measuring complex social science concepts at scale.

Authors' Note

The paper meets all ethical guidelines required by the University of Connecticut and it is not under consideration for publication outside of *JEBS*.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

Kylie L. Anglin https://orcid.org/0000-0001-7661-3370

References

Anglin, K. L., Boguslav, A., & Hall, T. (2022). Improving the science of annotation for natural language processing: The use of the single-case study for piloting annotation projects. *Journal of Data Science*, *20*(3), 339–357. https://doi.org/10.6339/22-JDS1054

Baker, R. S., & Hawn, A. (2021). Algorithmic bias in education. *International Journal of Artificial Intelligence in Education*, *32*, 1052–1092. https://doi.org/10.1007/s40593-021-00285-9

Boguslav, A. (2024). *Parsing coaching practice: A systematic framework for describing coaching discourse. AERA Open*, *10*. https://doi.org/10.1177/23328584241263861

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., & Brunskill, E. (2021). *On the opportunities and risks of foundation models*. arXiv Preprint arXiv:2108.07258.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., & Askell, A. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901. https://doi.org/10.48550/arXiv.2005.14165

Can, D., Marín, R. A., Georgiou, P. G., Imel, Z. E., Atkins, D. C., & Narayanan, S. S. (2016). ''It sounds like …'': A natural language processing approach to detecting counselor reflections in motivational interviewing. *Journal of Counseling Psychology*, *63*(3), 343. https://doi.org/10.1037/cou0000111

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Cohen, J., Wong, V., Krishnamachari, A., & Berlin, R. (2020). Teacher coaching in a simulated environment. *Educational Evaluation and Policy Analysis*, *42*(2), 208–231. https://doi.org/10.3102/0162373720906217

Deng, L., & Liu, Y. (2018). *Deep learning in natural language processing*. Springer.

Derry, A., Krzywinski, M., & Altman, N. (2023). Neural networks primer. *Nature Methods*, *20*(2), 165–167. https://doi.org/10.1038/s41592-022-01747-1

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *Bert: Pre-training of deep bidirectional transformers for language understanding*. arXiv Preprint arXiv:1810.04805.

Fernández, A., Garcia, S., Herrera, F., & Chawla, N. V. (2018). SMOTE for learning from imbalanced data: Progress and challenges, marking the 15-year anniversary. *Journal of Artificial Intelligence Research*, *61*, 863–905. https://doi.org/10.1613/jair.1.11192

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

Jain, A. K., Mao, J., & Mohiuddin, K. M. (1996). Artificial neural networks: A tutorial. *Computer*, *29*(3), 31–44. https://doi.org/doi.org/10.1109/2.485891

Lucy, L., Demszky, D., Bromley, P., & Jurafsky, D. (2020). Content analysis of textbooks via natural language processing: Findings on gender, race, and ethnicity in Texas U.S. History Textbooks. *AERA Open*, *6*(3), 1–27. https://doi.org/10.1177/2332858420940312

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing* (p. 680). MIT Press. https://dl.acm.org/citation.cfm?id=311445

Mcfarland, D. A., Khanna, S., Domingue, B. W., & Pardos, Z. A. (2021). Education data science: Past, present, future. *AERA Open*, *7*(1), 1–12. https://doi.org/10.1177/23328584211052055

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space*. https://doi.org/10.48550/arXiv.1301.3781

Mitchell, M., Wu, S., Zaldivar, A., Barnes, P., Vasserman, L., Hutchinson, B., Spitzer, E., Raji, I. D., & Gebru, T. (2019). *Model cards for model reporting. Proceedings of the Conference on Fairness, Accountability, and Transparency*. https://doi.org/10.1145/3287560.3287596

Mosbach, M., Andriushchenko, M., & Klakow, D. (2020). *On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines*. arXiv Preprint arXiv:2006.04884.

Nayak, P. (2019, October 25). *Understanding searches better than ever before*. Google. https://blog.google/products/search/search-language-understanding-bert/

OpenAI. (2023). *Enterprise privacy*. https://openai.com/enterprise-privacy

OpenAI. (2024a). *Fine-tuning*. https://platform.openai.com

OpenAI. (2024b). *Models*. https://platform.openai.com

OpenAI. (2024c). *OpenAI Platform*. Libraries. https://platform.openai.com/docs/libraries/python-library

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., & Ray, A. (2022). Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, *35*, 27730–27744. https://doi.org/10.48550/arXiv.2203.02155

Plaza-del-Arco, F. M., Martín-Valdivia, M.-T., & Klinger, R. (2022). *Natural language inference prompts for zero-shot emotion classification in text across corpora*. arXiv Preprint arXiv:2209.06701.

Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). *Improving language understanding by generative pre-training*. Preprint. https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, *1*(8), 9.

Ray, P. P. (2023). ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, *3*, 121–154. https://doi.org/10.1016/j.iotcps.2023.04.003

Reynolds, L., & McDonell, K. (2021). *Prompt programming for large language models: Beyond the few-shot paradigm*. Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1–7. https://doi.org/10.48550/arXiv.2102.07350

Savoy, J. (1997). Statistical inference in retrieval effectiveness evaluation. *Information Processing & Management*, *33*(4), 495–512. https://doi.org/10.1016/S0306-4573(97)00027-7

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms*. arXiv Preprint arXiv:1707.06347. https://doi.org/10.48550/arXiv.1707.06347

Sennrich, R., Haddow, B., & Birch, A. (2015). *Neural machine translation of rare words with subword units*. arXiv Preprint arXiv:1508.07909. https://doi.org/10.48550/arXiv.1508.07909

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*, 1–11. https://arxiv.org/abs/1706.03762.

Yogatama, D., Dyer, C., Ling, W., & Blunsom, P. (2017). *Generative and discriminative text classification with recurrent neural networks*. arXiv Preprint arXiv:1703.01898.

Authors

KYLIE L. ANGLIN is an assistant professor at the University of Connecticut, Storrs, CT, USA. Her research develops data science and natural language processing-based methods for understanding educational processes, alongside the development of methods for improving the validity and replicability of research.

CLAUDIA VENTURA is a graduate student at the University of Connecticut, Storrs, CT, USA. Her research interests include psychometrics and machine learning.

Manuscript received March 8, 2024
Revision received August 8, 2024
Accepted August 13, 2024