International Journal of Education and Social Science; Vol. 11 No. 1; April 2024
ISSN 2410-5171 (Online), 2415-1246 (Print)
Published by Research Institute for Progression of Knowledge

Using GPT-3 for Traditional Machine Learning: A Tutorial

David Rosenthal
Associate Professor of Computing and Decision Sciences
Stillman School of Business, Seton Hall University
400 South Orange Avenue, South Orange, NJ, 07079, USA.

Rob Weitz
Associate Professor of Computing and Decision Sciences
Stillman School of Business, Seton Hall University
400 South Orange Avenue, South Orange, NJ, 07079, USA.

Viswa Viswanathan
Professor of Computing and Decision Sciences
Stillman School of Business, Seton Hall University
400 South Orange Avenue, South Orange, NJ, 07079, USA.

Abstract

Large Language Models (LLM) like ChatGPT, Bard and Claude have clearly demonstrated that Artificial Intelligence (AI) has not just come of age but is fueling speculation that AI programs understand language like humans do and are even conscious. Machine Learning (ML) has been in the limelight longer. This tutorial looks at the intersection of these two exciting areas. It takes an early, high-performing LLM (GPT-3) outside the zone of its mainstream application and shows the steps to use it for more conventional ML tasks. We provide a quick and simple introduction to GPT-3 and ML and then show all the required steps from obtaining a GPT-3 account, all the way to running ML with GPT-3 on an example dataset. We also provide all the resources for people to try these steps on their own datasets. Some familiarity with running Python programs on a command line will help, but is not essential. Both Windows and Mac users can follow the tutorial.

Keywords: GPT-3, machine learning, tutorial

Introduction

The field of Artificial Intelligence (AI) aims to mimic human intelligence in many areas. Large Language Models (LLM) like GPT-3 (Generative Pre-Trained Transformer 3), and more recently GPT-3.5, GPT-4, and ChatGPT (Jovanovic, M. & Campbell, M. 2022) have bedazzled the public with their ability to generate high quality natural language text in response to prompts.

LLMs can assist us in any task involving generating text in response to a prompt. Some examples of such text include natural language, computer code, and code to supply as input to image generation programs. Surprisingly, LLM's seem to be powerful enough to apply to a subset of traditional ML problems as well. This tutorial focuses only on this aspect.

GPT-3 was trained on nearly a trillion words of text, including the content of the World Wide Web, numerous books, and other documents. In addition, we can further improve its performance by providing relevant examples for our specific problem. The creators of GPT-3 refer to this as "fine-tuning." This tutorial addresses fine-tuning GPT-3 for ML. At the time of writing, only GPT-3 allows fine-tuning (not the later versions).

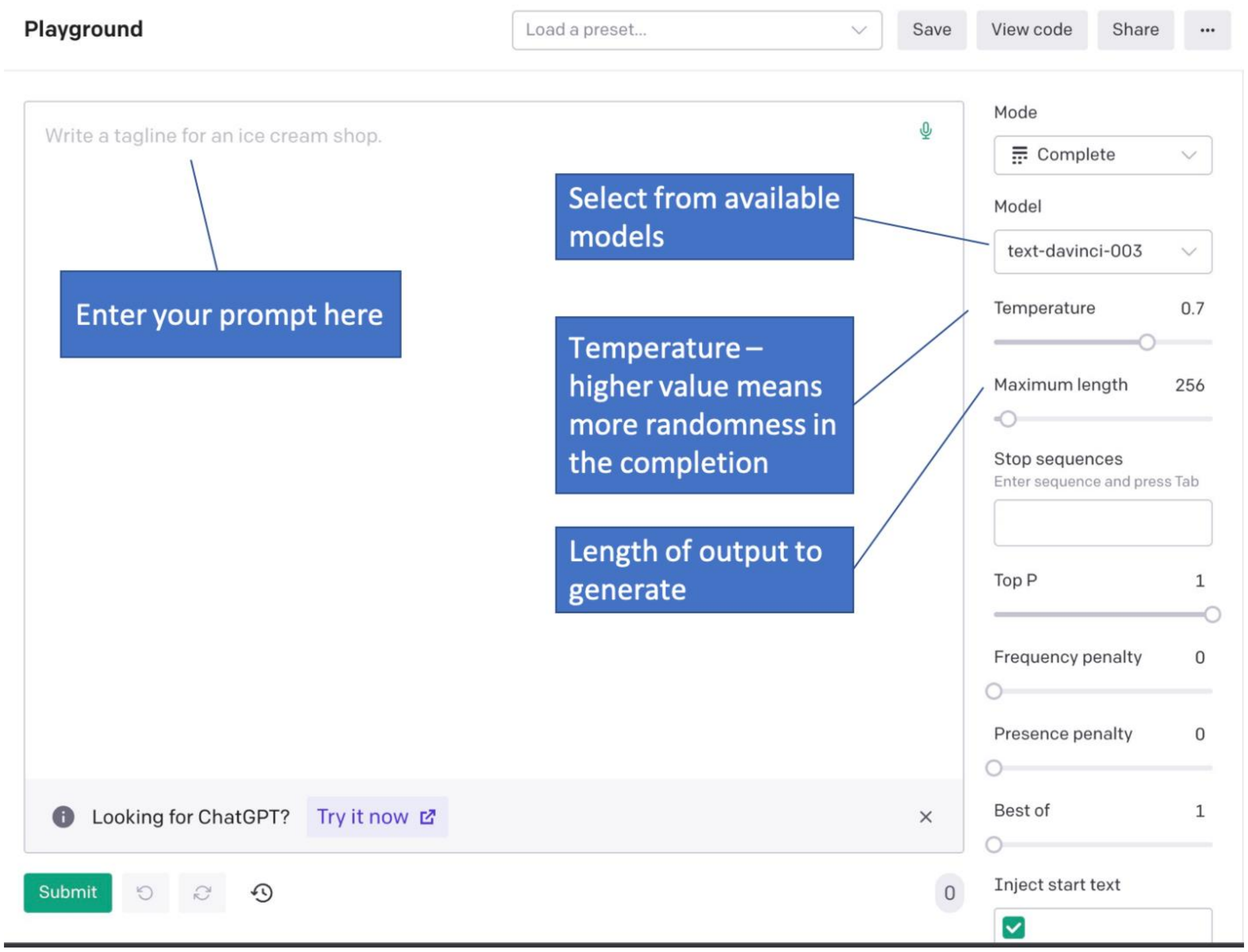
We briefly introduce GPT-3 and ML, and then show how we can use GPT-3 for ML. We also compare the performance of GPT-3 against traditional ML models. The tutorial does not go into the internal details of GPT-3.

What is GPT-3?

GPT-3 is a language prediction model. In response to *prompts*, it generates responses (*completions*) that seem indistinguishable from human responses. Being a large neural network capable of learning, it can also be fed additional data in a specialized form and we can use this feature to use it to perform traditional ML tasks.

Using GPT-3 Interactively

Later in this tutorial, we provide instructions for obtaining a GPT-3 account. Once you get the account, you can use GPT-3 interactively via a web browser.
Figure 1 shows its web user interface or the GPT-3 *Playground*. You can enter a *prompt* and click *Submit* to see GPT-3's response. GPT-3 provides several different base models, and we can select one from under the *Model* dropdown in the *Playground*. These differ, for example, in their speed and cost. Given this tutorial's purpose, it does not go further into the different models, or use GPT-3 interactively.


Enter your prompt here

Select from available models

Temperature – higher value means more randomness in the completion

Length of output to generate

Mode: Complete
Model: text-davinci-003
Temperature 0.7
Maximum length 256
Stop sequences
Top P 1
Frequency penalty 0
Presence penalty 0
Best of 1
Inject start text ✓

Figure 1: User interface of GPT-3

A QUICK INTRODUCTION INTO MACHINE LEARNING

Error! Reference source not found. shows a slice of data from a diabetes dataset (Kaggle.com) which many ML studies use as a benchmark (Fregoso-Aparicio, L. et al. 2021). The first eight columns show data items, including medical measurements of a set of non-diabetic people. The researchers who gathered the data returned to the same people after six months and found out if each of them had developed diabetes and added the data in the column *(Outcome:* 0: Non-diabetic, 1: Diabetic).

Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	Diabetes Pedigree Function	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1
1	85	66	29	0	26.6	0.351	31	0
8	183	64	0	0	23.3	0.672	32	1
1	89	66	23	94	28.1	0.167	21	0
0	137	40	35	168	43.1	2.288	33	1
5	116	74	0	0	25.6	0.201	30	0
3	78	50	32	88	31.0	0.248	26	1

Figure2: Slice of the diabetes data

Let us assume that there is an underlying relationship between the values in the first eight columns (predictor attributes) and the value in the last column (target attribute). Let us also assume that we can learn this relationship in the form of a model. We can then apply the model to data from new people outside of the dataset that we used to build the model and predict whether each of them will develop diabetes within six months. From a practical viewpoint, the model can help in early identification of people at risk and take preventive action.

In ML, we leave the task of "learning the relationship" to a machine (computer) by supplying it with the examples in the dataset.

Figure 2 depicts the process of building a ML model based on examples.

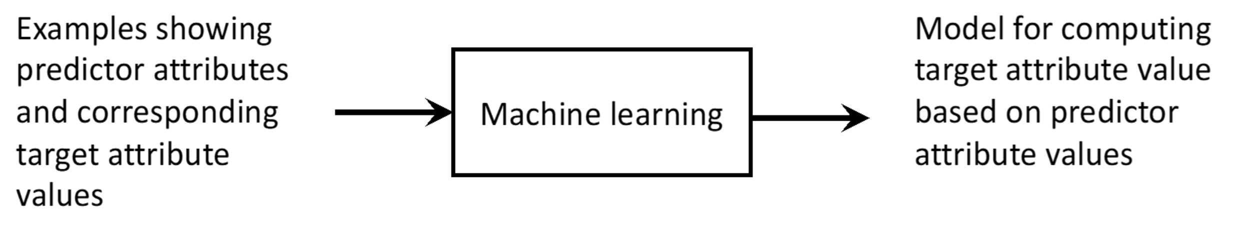

Examples showing predictor attributes and corresponding target attribute values
Machine learning
Model for computing target attribute value based on predictor attribute values

Figure 2: Machine learns the relationship between predictor and target attributes (model)

Figure 3 shows how the model is applied on a new case to compute the target attribute value given the predictor attributes as inputs.

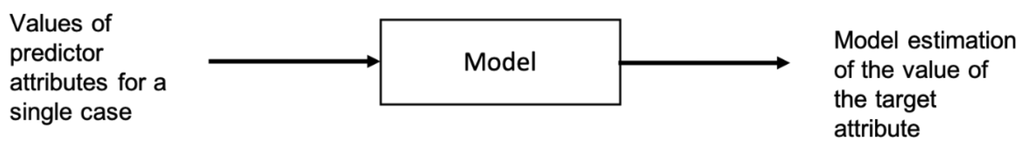

Values of predictor attributes for a single case
Model
Model estimation of the value of the target attribute

Figure 3: Model being used to compute the target attribute value based on predictor attribute values.

In our diabetes example, the target attribute is *categorical* – that is, non-numeric. The model predicts the value of the target attribute – here *Outcome*. As such, it represents a *classification* problem. ML can also be used for problems where the target attribute is numeric – namely *regression* models. This tutorial only considers classification.

We must assess the quality of a model before deploying it for use in the field. To get an unbiased assessment of the performance of a model on new data (data that it was not trained on), we randomly partition the original data into *training* and *test* partitions. We then use only the *training* partition for learning. The *test* set remains unseen by the ML process and is used only when we evaluate the model.

Once we have built a model using only the training data, we then use it to generate predictions for each row of the test partition. Since the test partition was generated from the original dataset, it already contains the correct values for the target attribute. We can therefore compare the model predictions against the known correct values to evaluate the model (Figure 5).

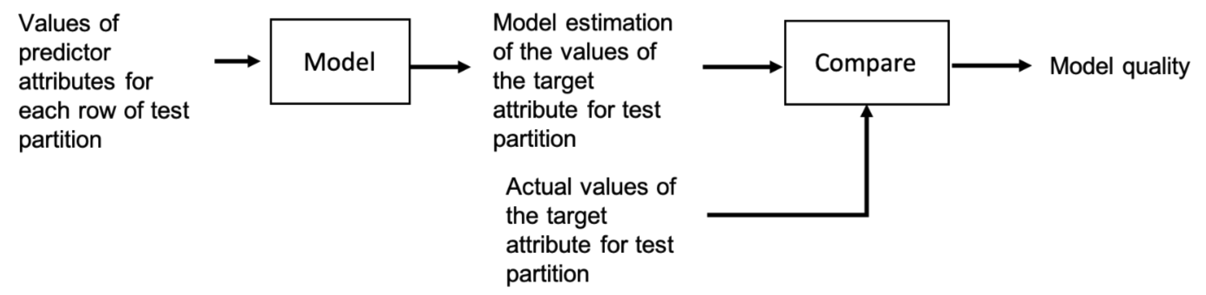

Values of predictor attributes for each row of test partition
Model
Model estimation of the values of the target attribute for test partition
Compare
Model quality
Actual values of the target attribute for test partition

Figure 4: Model evaluation on the test partition

Figure 6 shows a comprehensive view of the model building, evaluation, and deployment phases.

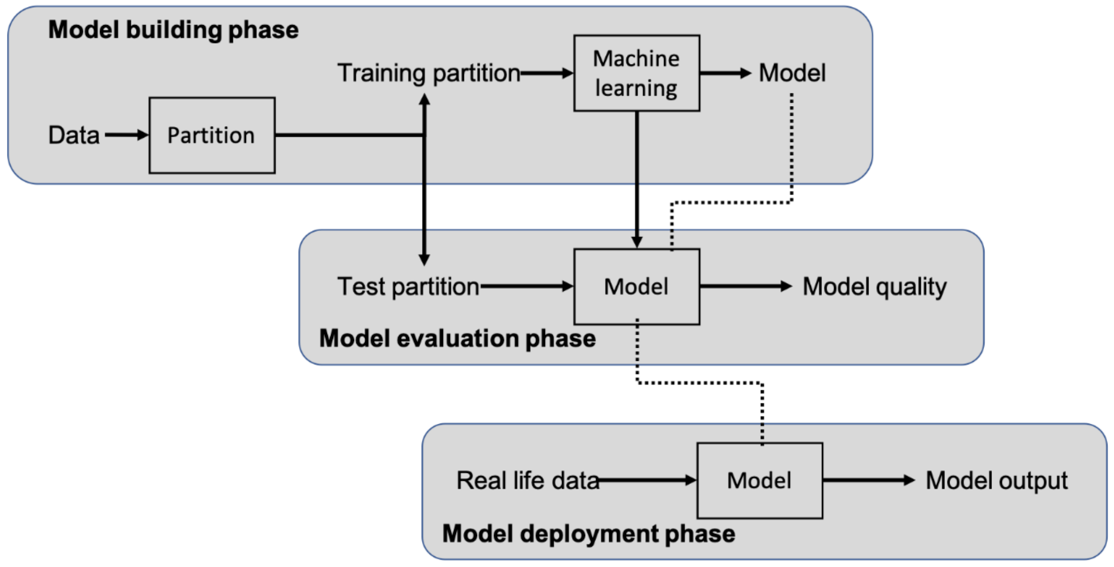


Figure 5: Model building, model evaluation and model deployment in ML

Other examples of classification tasks amenable to ML include:

- Using data about past credit card transactions as examples to build a model to identify whether a new transaction is likely fraudulent.
- Based on historical data with demographic information on many people and whether they bought a product, build a model for identifying whether a new person will likely buy a product.
- Based on digitized image data from many genuine and fake paintings, build a model to identify whether a given new painting is genuine.

Methods for building classification models include trees, support vector machines, naïve bayes, logistic regression, k-nearest neighbors, and neural networks.

As a general AI system that consumes a prompt and produces a completion, GPT-3 can also mimic the functionality of traditional ML classifiers.

FINE TUNING GPT-3 FOR CLASSIFICATION PROBLEMS

We can use GPT-3 interactively for classification through its web interface by typing in our *prompt* (comprising the values of the predictor attributes) and getting back its *completion* (predicted value of the target attribute). We can also submit a prompt and receive a completion by accessing GPT-3 through a computer program, commonly a Python program.

At the simplest level, we submit a prompt for a single row to GPT-3 (interactively or programmatically) and receive a completion. However, before submitting our prompt, we can also give it some examples of prompts and correct completions to give some hints to GPT-3 about our specific domain. We then submit our new prompts. GPT-3 performs significantly better even when given a few examples.

We can enhance GPT-3's performance significantly by providing many examples (hundreds or even thousands), but this can be cumbersome to perform interactively. Furthermore, if we want to use the same set of examples across multiple sessions, then we would need to supply them repeatedly.

Fortunately, through a process called *fine-tuning*, GPT-3 allows us to submit many examples to augment a base model and then save the fine-tuned model. We can then use this fine-tuned model repeatedly without the need to supply examples. In this section we show the steps for fine-tuning GPT-3 for ML. (For more details see https://platform.openai.com/docs/guides/fine-tuning.)

Setting up

Get an OpenAI account**:** You can sign up for an account at www.openai.com/api. Approval could take some time. Getting a GPT-4 account will give you access to GPT-3 as well. Currently, GTP-4 does not support fine-tuning.
Get an API secret key. In addition, you need a (free) Application Programming Interface (API) secret key to access GPT-3 programmatically. Get it by logging into your account and then visiting https://platform.openai.com/account/api-keys and clicking on *Create new secret key*. You should then view and copy the text of the key into a text file for later use.

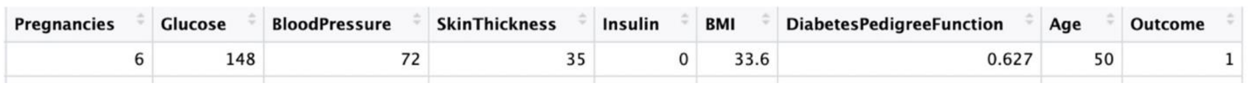
Install Python**.** If you have not already installed Python, download it from https://www.python.org/downloads/ for your operating system. Python comes with IDLE, a basic development environment. Although we explain the process with IDLE/Python, you can use any environment to run the code for this tutorial.

Prepare your dataset

For fine-tuning, GPT-3 needs examples in the form of *prompts* and corresponding *completions.* We need to first convert our tabular data suitably.

Converting tabular data to prompt-completion format

Figure 7 shows one row of a typical tabular dataset, often represented as a Comma Separated Values (csv) file.



Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
6	148	72	35	0	33.6	0.627	50	1

Figure 6: First example from our dataset

Figure 8 shows how the single example shown in Figure 7 can be converted into a prompt-completion pair represented in *json* format. If you look closely at Figure 8 you will see that it is formatted as
{“prompt”:”<predictor values in example>”, “completion”:”<target attribute value>”}

```
{"prompt":"Pregnancies:6\nGlucose:148\nBloodPressure:72\nSkinThickness:35\Insulin:0\nBMI:
33.6\nDiabetesPedigreeFunction:0.627\nAge:50\nCategory:","completion":"1"}
```

Figure 7: First row of data converted into prompt-completion format in json format that we can use to fine-tune GPT-3

In a similar vein, we can convert an entire dataset into the *prompt-completion* format.
We provide a utility program in Python which takes a dataset in csv format and converts it into the prompt-completion format ready to be used with GPT-3.

Preparing the diabetes dataset for GPT-3

For this tutorial we use the diabetes dataset already discussed. Note that the data set here is slightly different from the original, which has some missing values. ML practitioners use various methods to handle missing values. We imputed approximate values using the cart method in the MICE package in R (van Buuren, S., & Groothuis-Oudshoorn, K. 2021, Van Buuren, S., et al. 2023).

We have already partitioned the data and you can download the training and test partitions (diabetes-train.csv and diabetes-test.csv) from the resources associated with this article. We also converted the training set into the json format as in
Figure 7 and this file is available as diabetes-train.json. You will need diabetes-train.json and diabetes-test.csv to complete this tutorial. (We provide the two training csv files as well for those who want to attempt other classification methods on this data. The resources include the Python code CreateTrainingAndTestingfiles.py for partitioning any dataset in csv format.

You are now ready to start fine tuning.

Install OpenAI

After successfully installing Python, use the following command from a command prompt (Windows) or Terminal (Mac or Unix) while connected to the Internet:

```
pip install --upgrade openai
```

Note: If your system responds that it doesn’t recognize pip, it is likely that the folder containing pip is not in your system’s path. pip is part of the Python installation and should be in c:\....Programs\Python\Pythonxxx\Scripts, where xxx is the Python version you downloaded.(For example -- C:\Users\username\AppData\Local\Programs\Python\Python311\Scripts)

The pip install command installs the Python package openai, which contains functions to interact programmatically with GPT-3.

Setting your openai secret key in the environment

Before accessing openai programmatically, you need to establish your secret key as an operating system environment variable. Run the following command from a Windows command prompt or Mac terminal:

```
set OPENAI_API_KEY=<open api key>
```

where <open api key> is the text of the secret key that you saved soon after creation.

Creating a GPT-3 model fine-tuned on the training data

If using Windows, enter the following command in a command prompt (our training json file is in c:\temp-GPT\diabetes-train.json)

```
openai api fine_tunes.create -t c:\temp-GPT\diabetes-train.json -m ada
```

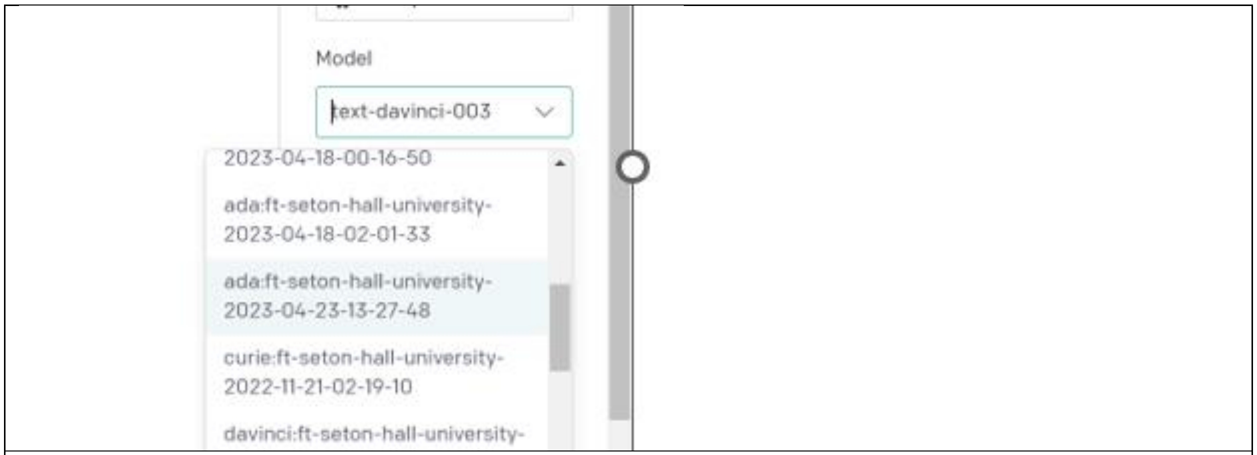
If using Mac, change the file path suitably in the above command.
Above we are fine-tuning GPT-3's *ada* model. In its place you could use *babbage*, *curie* or *davinci*. We're using *ada* as an example. You should get a response like this:

```
Upload progress:
100%|████████████████████████████████████████| 89.9k/89.9k [00:00<?, ?it/s]
Uploaded file from c:\temp-GPT\diabetes-train.json: file-pRrYjmHErnXICxBtDK0gd8WJ
Created fine-tune: ft-qjYP1aah3tdkJDp5lZ1dO3v2
Streaming events until fine-tuning is complete...

(Ctrl-C will interrupt the stream, but not cancel the fine-tune)
[2023-07-15 22:45:22] Created fine-tune: ft-qjYP1aah3tdkJDp5lZ1dO3v2

Stream interrupted (client disconnected).
To resume the stream, run:

  openai api fine_tunes.follow -i ft-qjYP1aah3tdkJDp5lZ1dO3v2
```

Fine-tuning can take up to 30 minutes, and possibly longer. Once the process finishes, you can get the name of the fine-tuned model from the Playground. If you are logged into the Playground, logout and log in again and get the name of the fine-tuned model from the *model* dropdown. (You may have to do this multiple times for the Playground to refresh and show the new model.)

OpenAI is in the process of updating GPT-3, as well as revising GPT-3.5 and GPT-4 to allow for fine-tuning. As of this writing, the following instructions are correct but there may be slight differences when you view the Playground.

Start by clicking in the Mode field and select Complete Legacy. In the model field there will be a standard model name (currently "test-davinci-003)." Clicking in this field will yield a pop-up window listing the fine-tuned model(s) you've created. The format is model name followed by your organization name followed by the date and time. See Figure 9.



Model
text-davinci-003

2023-04-18-00-16-50
ada:ft-seton-hall-university-2023-04-18-02-01-33
ada:ft-seton-hall-university-2023-04-23-13-27-48
curie:ft-seton-hall-university-2022-11-21-02-19-10
davinci:ft-seton-hall-university-

Figure 9: Viewing the models you've created.

<u>Using the fine-tuned model to make predictions for the test data</u>
You can use the Python program RunDataThroughGPT-3.py to generate the fine-tuned model's predictions for all the cases in a test set, and to compare them with the actual results.

You will need to edit the code to enter the name of your fine-tuned model. To do this, open IDLE (the interactive development environment that comes with Python. (You may of course use a different IDE or simply use Notepad or equivalent text editor.) In IDLE choose File/Open to open RunDataThroughGPT-3.py. You should see the code in the IDLE window. Insert your model name between the double quotation marks in the statement:

```
model = "",
```

and save the file.

With this done, you can run the code at the command line.

```
python RunDataThroughGPT-3.py
```

Alternatively, you can run the program in IDLE. In this case, however, you will have to make an additional change to the code. Running the code from IDLE requires that the code includes your openai api key (even if you entered it previously at the command prompt). So you should also uncomment (remove the leading #) and edit the line:

```
openai.api_key = ""
```

inserting your openai key between the double quotation marks.

To run the code, click on Run/Run.

The program will bring up the standard file dialog box asking for the test csv file name. The Python script reads each line from the test csv file, converts it into a prompt and submits it to the model. It obtains the completions provided by the fine-tuned model and compares the model’s predictions to the correct values in the test dataset. Figure 10 shows the error matrix from the diabetes test dataset. Your results should be similar, but not necessarily exactly the same. It will probably take a minute or two for your results to appear.

```
c:\Temp-GPT>python RunDataThroughGPT-3.py
Select testing file: diabetes-test.csv
Error Matrix
             Predicted Outcomes
                     1        0
Actual       1       58       22
Outcomes     0       36       114
```

Figure 10: GPT-3 Confusion Matrix

Here the rows represent the actual outcomes, and the columns are the predicted outcomes.

RESULTS AND COMPARISON

The values in the top left and bottom right in Figure 10 are correct predictions. Eighty of the subjects in the test set have diabetes; the model correctly predicted 58 of them and misclassified 22 of them (as not being diabetic). Likewise, 150 subjects in the test set did not have diabetes; the model correctly predicted 114 of them and misclassified 36 of them.

This provides an overall accuracy rate of (58 + 114)/(58 + 22 + 36 + 114) = 74.8%.

In classification problems, in addition to the overall accuracy, we are often interested in how the model performs on each of the classes. We might be more interested in the model’s performance on one of the classes. The class of interest is the positive class. In our current example, we might be more interested when a person is predicted as likely to develop diabetes rather than when the model predicts that a person will not develop diabetes. Hence, an outcome of 1 will be our positive class.

Two additional commonly reported summary measures are *sensitivity* and *specificity*, where sensitivity is defined as the probability of predicting a positive outcome given an actual positive and specificity is the probability of predicting a negative outcome given an actual negative. These calculations are below:

Sensitivity = 58/80 = 72.5%
Specificity = 114/150 = 76.0%

Although this tutorial does not aim to compare GPT-3 with other classifier methods, we show comparisons to provide context. We used a logistic regression model on the same training and test data; the summary results follow.

Accuracy: 0.743
Sensitivity: 0.563
Specificity: 0.840

These results are in line with those published in the literature where similar data preprocessing and classification techniques were used. See Albahli S.(2020), table 4, p. 1073 for example.

Steps to perform classification with GPT-3 on your own dataset

Unless your data is already partitioned into a json training set and csv test set, you will need to do that first. You can use our Python program CreateTrainingAndTestingFiles.py. Input to the program is your data set in csv format. The file should have column headings, and the last column should be the outcome variable. The only other requirement is that the value of the outcome variable must be 0 or 1. Typically, the outcome of interest is designated as 1.

To run the program, enter python CreateTrainingAndTestingFiles.py at the command prompt. (This assumes the file is in the current directory or in the path.)

Alternatively, you can run the program in IDLE. In IDLE choose File/Open to open CreateTrainingAndTestingFiles.py. You should see the code in the IDLE window. To run the code, click on Run/Run.

The program will first prompt you for the percentage of data you want in your training set (an integer between 0 and 100). After you hit Enter, it will then open a file dialog box from which you select the csv file containing your data. The partitioning program and the input csv file do not need to be in the same folder; the training and test csv files will appear in the same folder as the input data.

The training and test records are randomly selected, but you will get the same partition each time you run the program on the same input data. If you want a different partition, edit the two set.seed(0) statements, replacing the 0 with a different number.

Output is three files: *filename*-train.json, *filename*-train.csv, and *filename*-test.csv, where *filename* is the name of your initial csv file. (Again, for running GPT-3 as a classifier, you will only need the training json file and the test csv file. The training csv file is provided if you want to run a traditional classification method where a csv file is likely what you'll need.)

You may then go through the steps outlined in this paper, substituting your training and test files for those specified here.

Conclusion

We have provided a step-by-step tutorial demonstrating fine tuning using GPT-3 for classification.

A major objective of this paper is to provide an opportunity for the reader to explore fine tuning in GPT-3 with relatively low effort, using a well-known and easily understood class of problems. Additionally, the paper provides the reader with resources to use GPT-3 for ML using their own data.

Further, we've demonstrated that, at least for the benchmark data set used here, GPT-3 performs in the range of traditional classification methods.

References

Jovanovic, M. & Campbell, M. (2022) "Generative Artificial Intelligence: Trends and Prospects" in Computer, vol. 55, no. 10, pp. 107-112.

Kaggle.com. (2023) "Pima Indians Diabetes Database. https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database. Accessed 16 July 2023]

Fregoso-Aparicio L, Noguez J, Montesinos L, & García-García JA. (2021) ML and deep learning predictive models for type 2 diabetes: a systematic review. Diabetology & Metabolic Syndrome. 2021 Dec 20;13(1):148.

van Buuren, S., & Groothuis-Oudshoorn, K. (2011). mice: Multivariate Imputation by Chained Equations in R. Journal of Statistical Software, 45(3), 1–67.

Van Buuren, S., et al. (2023) MICE. https://cran.r-project.org/web/packages/mice/mice.pdf. Accessed 16 July 2023.

Albahli S. (2020) Type 2 ML: an effective hybrid prediction model for early type 2 diabetes detection. J Med Imaging Health Inform. 10(5):1069–75.