

 Edward-Kuhn / ml_Assaginment Public

generated from [Thayer-ENG5108/Assignment_5_Fall2022](#)

<> Code

Issues

Pull requests

Actions

Projects

Wiki


Security


Insights

 main ▾




ml_Assaginment / Assignment_6_Fall2022.Dianhao_Liu.ipynb

 Edward-Kuhn Add files via upload History

 1 contributor

2968 lines (2968 sloc) | 114 KB



ENGS 108 Fall 2022 Assignment 6

Due November 11, 2022 at 11:59PM on Github

Instructors: George Cybenko

TAs: Chase Yakaboski and Clement Nyanhongo

Rules and Requirements

1. You are only allowed to use Python packages that are explicitly imported in the assignment notebook or are standard (builtin) python libraries like random, os, sys, etc. (Standard Builtin Python libraries will have a Python.org documentation). For this assignment you may use:

- [numpy](#)
- [pandas](#)
- [scikit-learn](#)
- [matplotlib](#)
- [tensorflow](#)

2. All code must be fit into the designated code or text blocks in the assignment notebook. They are identified by a **TODO** qualifier.

3. For analytical questions that don't require code, type your answer cleanly in Markdown. For help, see the [Google Colab Markdown Guide](#).

```
In [1]: ''' Import Statements '''
import os
import random
import numpy as np
import pandas as pd
import csv
import tqdm
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
Out[1]: True
```

Creating a Trump Tweet Generator!!!

Finally it's happened. I am allowed to make my dream assignment. Trust me it's the best assignment. People tell me all the time how they wish they could have had an assignment as great as this. You will see why! But first some housekeeping...

Data Loading

In this assignment we will be using a repository of Donald Trump tweets scrapped from Twitter through June 2020 from [Kaggle](#) and will use the following code blocks to download the dataset directly to your Google Drive.

Creating a Kaggle API Token

First we will need to download an API token from Kaggle in order to download the dataset, so our first step is to create a Kaggle account if you don't already have one. (You should have done this in Assignment 4 in case this looks familiar.

1. Create a Kaggle account by following the sign up instructions [here](#).
2. Log into your Kaggle account and click your account icon on the upper righthand side.
3. Then select **Account** from the dropdown/sidebar menu.
4. Scroll down to the **API** section and select **Create New API Token**.
5. This will download a JSON file called kaggle.json to your Downloads folder on your computer.
6. Now run the following code block and when the **Browse** button appears, click it and select that kaggle.json file.

```
In [2]: # Run this code block after creating a Kaggle API token as instructed above
! pip install -q kaggle
from google.colab import files

# Will ask you to upload kaggle.json file and remove any old ones.
if os.path.exists('kaggle.json'):
    os.remove('kaggle.json')
files.upload()

# Will create the appropriate directory structure
if not os.path.exists('/root/.kaggle'):
    ! mkdir ~/.kaggle
    ! cp kaggle.json ~/.kaggle/
    ! chmod 600 ~/.kaggle/kaggle.json
# Also we are going to make a directory called result
if not os.path.exists('/content/results'):
    ! mkdir /content/result
```

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving kaggle.json to kaggle.json

Downloading the Dataset

7. Now we have downloaded our Kaggle credentials we can now download the Trump Tweets Dataset (or any other Kaggle dataset for that matter) directly into our Google Drive.

```
In [3]: ! kaggle datasets download austinreese/trump-tweets
# Will check to see if the yoga postures zip file has been unzipped and will
! unzip trump-tweets.zip
```

Downloading trump-tweets.zip to /content
100% 6.88M/6.88M [00:01<00:00, 8.72MB/s]
100% 6.88M/6.88M [00:01<00:00, 5.37MB/s]
Archive: trump-tweets.zip
 inflating: realdonaldtrump.csv
 inflating: trumptweets.csv

Loading the Dataset using Pandas

Now let's inspect the trump tweets dataset and see what we have to work with...
Brace yourselves.

```
In [14]: # Let's load in the two files that we inflated from the Kaggle download. Both
real_donald_trump_df = pd.read_csv('realdonaldtrump.csv')
```

```
In [5]: real_donald_trump_df.head()
```

Out[5]:

	id	link	content	date	retwee
0	1698308935	https://twitter.com/realDonaldTrump/status/169...	Be sure to tune in and watch Donald Trump on L...	2009-05-04 13:54:25	51
1	1701461182	https://twitter.com/realDonaldTrump/status/170...	Donald Trump will be appearing on The View tom...	2009-05-04 20:00:10	3
2	1737479987	https://twitter.com/realDonaldTrump/status/173...	Donald Trump reads Top Ten Financial Tips on L...	2009-05-08 08:38:08	1
3	1741160716	https://twitter.com/realDonaldTrump/status/174...	New Blog Post: Celebrity	2009-05-08	1

Apprentice 15:40:15
Finale and...

		"My		
		persona	2009-	
4	1773561338	https://twitter.com/realDonaldTrump/status/177...	will never	05-12 137
			be that of a	09:07:28
			wallflower...	

Problem 1: Pre-Processing the Tweets

As you may have noticed from the dataframe we just loaded, there are some special characters we need to handle. If we are making a tweet or sentence generator, we don't want to mess with special characters like commas or colons or really even captialization. So in the following section you are going to preprocess the data and strip these special characters out.

Task 1: The Preprocess function

In the following code block complete the preprocess function that will strip or substitute out various special characters or sequences of characters that may not be ideal when training a sentence generator. We will use the built-in re python library to do a number of substitutions, and I have given you a skeleton below, see if you can complete it.

In [7]:

```
import re

REGEX_SUBS = {
    r'\\': ' ',
    r'\n': ' ',
    r'&': ' ',
    r'RT ': ' ',
    r'~': ' ',
    r'#': ' ',
    r'!+': ' ',
    r'(http[s]?[S+])|(\w+\.[A-Za-z]{2,4}\S*)': 'link',
    r'*': ' ',
    r'[@]\w+': 'user',
    r':|;': ' ',
    r'\\x\\w+|\\d': ' ',
    r'\\x\\w+|\\d': ' ',
    r'\\x\\w+': ' ',
    r' ': ' ',
    r' ': ' ',
    r' ': ' ',
}

def preprocess(text):
    # TODO: Complete the function and using the provided regular expression s
    for key, value in REGEX_SUBS.items():
        #TODO: Do something, maybe look at the re.sub command.
```

```
temp = re.sub(key, value, text)
#TODO: Also make everything lowercase
text = temp.lower()
return text
```

Task 2: Preprocess the dataset.

Now that we have our preprocess function, let's preprocess all the tweets.

```
In [15]: #TODO: Run all the Dataframe content through your preprocess function.
for i in range(len(real_donald_trump_df['content'])):
    real_donald_trump_df['content'][i] = preprocess(real_donald_trump_df['con
```

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

This is separate from the ipykernel package so we can avoid doing imports until

Task 3: Tokenize the Data

The next step in every Natural Language Processing task is to tokenize the data, i.e., separate our words, special characters, etc. into separate unique tokens. We will be using the [Natural Language Tool Kit](#) in python to accomplish this. Study the nltk API docs and see if you can tokenize our data.

```
In [17]: from nltk.tokenize import word_tokenize

tokenized = []
# TODO: Tokenize each preprocessed tweet
for i in range(len(real_donald_trump_df['content'])):
    tokenized.append(word_tokenize(real_donald_trump_df['content'][i]))
```

Task 4: Build a Vocabulary

Now that we have our data tokenized, let's build a vocabulary including beginning and ending of sentence tokens, i.e., -for example. At this point let's also add in these beginning and end tokens into each of our data instances.

```
In [25]: vocab_keys = {'.', ''}
#TODO: Build a vocabulary dictionary and inverse vocabulary dictionary
for i in range(len(tokenized)):
    vocab_keys.update(tokenized[i])
vocabulary = {token: idx for idx, token in enumerate(vocab_keys)}
vocabulary_inverse = {idx: token for token, idx in vocabulary.items()}

#TODO: Encode training data.
train_data = []
```

```

for i in range(len(tokenized)):
    for j in range(len(tokenized[i])):
        train_data.append(vocabulary.get(tokenized[i][j]))

```

Problem 1: N-gram Language Model

In this problem, you will be building a couple n-gram language modeling and see how well just taking pure frequency counts and building a conditional probability distribution will work.

Task 1: A 2-gram (Bigram) Model

Recall that our goal in building a language model is to represent the conditional probability $P(w_i|w_{i-1})$ for pairs of words w_i and w_j .

Part A: Frequencies

Go through the encoded Trump tweets and calculate the frequencies of all words as well as all pair of words that appear next to each other in the corpus.

```

In [27]:
from collections import defaultdict
from typing import Counter

def make_grams(tweets, n=2):
    #TODO: Complete the function to make the necessary bigrams.
    grams = list(nltk.bigrams(tweets))
    return grams

def calculate_counts(grams):
    #TODO: Complete the function to count all the bigrams.
    counts = Counter(grams)
    del counts[(vocabulary[''], vocabulary['-'])]
    return counts

```

```

In [28]:
grams = make_grams(train_data)
counts = calculate_counts(grams)

```

Part B: Probabilities

Now from the counts above we will calculate an associated conditional probabilities.

```

In [38]:
temp = Counter(train_data)
probs = defaultdict(dict)
for gram, count in counts.items():
    #TODO: Do something...
    probs[gram] = count / temp[gram[0]]

```

Part C: Make an Bigram Generator Function

Now that we have our probability distribution, let's make a generator function so that we can generate random Trump tweets using our bigram language model.

```
In [87]: def get_next_word(next_word_probs, last_word, something, n):
    #TODO: Write a function to get the next word based on the previous words

    for i in range(n):
        count = -1
        temp = 0

        for gram, count in next_word_probs.items():
            if last_word == gram[0] and count > temp:
                temp = count
                next_word = gram[1]
            count += 1

        if count == len(next_word_probs)-1:
            break

        if next_word != vocabulary[' ']:
            something.append(next_word)

        last_word = next_word
    return something

def generate(start_text='', n=10, probs=probs, vocabulary=vocabulary, vocab
    # Helper code to create the start text.
    start_text = [' '] + nltk.word_tokenize(preprocess(start_text))
    last_word = vocabulary[start_text[-1]]

    #TODO: Encode and generate a trump tweet.
    something = []
    next_word = []

    for i in start_text:
        next_word.append(vocabulary[i])

    next_word = get_next_word(probs, last_word, next_word, n)
    next_word.remove(vocabulary[' '])

    for i in next_word:
        something.append(vocabulary_inverse[i])

    something = " ".join(something)
    return something
```

```
In [88]: generate('I can make', n=4)
```

```
Out[88]: 'i can make america great again !'
```

(Bonus) Task 2: Make a Trigram Model

Using your code from Parts A-C, see if you can build a trigram (3-gram) model that might make tweets a little more logical.

```
In [ ]: #TODO: Your code goes here.
```

Problem 2: Using a Transformer

In this section we are going to leverage a pretrained transformer, i.e., [GPT-2](#), built by the [Hugging Faces Team](#). We will also be using their tokenizers because they have been optimized for the language generation task. You should be aware that behind the scenes, their model is using [PyTorch](#), the deep learning library built by Facebook, and is quickly becoming more popular than our beloved Tensorflow.

Task 1: Install and load the GPT2 Model

```
In [89]: # Run the following code to install the HuggingFace's transformer python pa  
! pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
```

```
Collecting transformers
```

```
  Downloading transformers-4.24.0-py3-none-any.whl (5.5 MB)
```

```
|████████████████████████████████████████| 5.5 MB 35.1 MB/s
```

```
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers) (4.64.1)
```

```
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from transformers) (21.3)
```

```
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (1.21.6)
```

```
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers) (3.8.0)
```

```
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.7/dist-packages (from transformers) (6.0)
```

```
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (2022.6.2)
```

```
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
```

```
  Downloading tokenizers-0.13.2-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.6 MB)
```

```
|████████████████████████████████████████| 7.6 MB 59.4 MB/s
```

```
Collecting huggingface-hub<1.0,>=0.10.0
```

```
  Downloading huggingface_hub-0.11.0-py3-none-any.whl (182 kB)
```

```
|████████████████████████████████████████| 182 kB 73.8 MB/s
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers) (2.23.0)
```

```
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from transformers) (4.13.0)
```

```
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7/dist-packages (from huggingface-hub<1.0,>=0.10.0->transformers) (4.1.1)
```

```
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0->transformers) (3.0.9)
```

Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->transformers) (3.10.0)
 Requirement already satisfied: urllib3!=1.25.0,!<1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (1.24.3)
 Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2.10)
 Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (3.0.4)
 Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2022.9.24)
 Installing collected packages: tokenizers, huggingface-hub, transformers
 Successfully installed huggingface-hub-0.11.0 tokenizers-0.13.2 transformers-4.24.0

```
In [93]: # TODO: Follow the Link above and Load the GPT2 model as well as the tokenizer
from transformers import GPT2Tokenizer, GPT2LMHeadModel
# NOTE: Replace GPT2Model with GPT2LMHeadModel (this is the model for finetuning)
model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
```

```
Downloading: 0%|          | 0.00/665 [00:00, ?B/s]
Downloading: 0%|          | 0.00/548M [00:00, ?B/s]
Downloading: 0%|          | 0.00/1.04M [00:00, ?B/s]
Downloading: 0%|          | 0.00/456k [00:00, ?B/s]
```

Task 2: Formatting our Training Data

Upcoming we will be fine-tuning this GPT-2 model on our Trump Tweets, but in order to leverage some of the utility classes built by HuggingFaces, we want to take our preprocessed Trump tweets and place them in a flat text file.

```
In [94]: #TODO: Take our preprocessed trump tweets and write them to a text file
with open('trumpdata.txt', 'w') as text_file:
    #TODO: Do something
    for i in real_donald_trump_df['content']:
        text_file.write(i)
```

Task 4: Fine-tuning the Model

In the following sections, we will complete a couple functions that will allow us to fine-tune the GPT-2 model to our Trump Tweets. I am going to give you a couple of these helper functions, but leave you to write parts of the training function. See the following documentation from Hugging Faces to see the attributes for the [Trainer Class](#).

```
In [102... from transformers import Trainer, TrainingArguments
from transformers import TextDataset, DataCollatorForLanguageModeling

def load_dataset(file_path, tokenizer, block_size = 128):
    # Will load and tokenize the data
    dataset = TextDataset(
```

```

dataset = TextDataset(
    tokenizer = tokenizer,
    file_path = file_path,
    block_size = block_size,
)
return dataset

def load_data_collator(tokenizer, mlm = False):
    # Helper Function
    data_collator = DataCollatorForLanguageModeling(
        tokenizer=tokenizer,
        mlm=mlm,
    )
    return data_collator

def train(
    train_file_path,
    model,
    tokenizer,
    output_dir,
    overwrite_output_dir,
    num_train_epochs,
    save_steps
):
    #TODO: Use the helper functions above to load the dataset and data collector
    train_dataset = load_dataset(train_file_path, tokenizer)
    data_collector = load_data_collator(tokenizer)

    # Don't mess with.
    tokenizer.save_pretrained(output_dir)
    model.save_pretrained(output_dir)

    # Don't mess with.
    training_args = TrainingArguments(
        output_dir=output_dir,
        overwrite_output_dir=overwrite_output_dir,
        per_device_train_batch_size=8,
        num_train_epochs=num_train_epochs,
    )
    #TODO: Use the Trainer class with the necessary parameters to instantiate
    trainer = Trainer(
        model = model,
        args = training_args,
        train_dataset = train_dataset,
        data_collator = data_collector
    )

    #TODO: Train and save the model using the train and save functions built
    trainer.train()
    trainer.save_model(output_dir)

```

In [96]:

```

#TODO: Set necessary parameters, here are some defaults.
train_file_path = "/content/trumpdata.txt"
output_dir = '/content/result'
overwrite_output_dir = False
num_train_epochs = 1
save_steps = 500

```

In [103]:

```
#TODO: Use your train function to train the model. It takes about 30 minutes
trained = train(
    train_file_path,
    model,
    tokenizer,
    output_dir,
    overwrite_output_dir,
    num_train_epochs,
    save_steps
)
```

/usr/local/lib/python3.7/dist-packages/transformers/data/datasets/language_modeling.py:58: FutureWarning: This dataset will be removed from the library soon, preprocessing should be handled with the 🤗 Datasets library. You can have a look at this example script for pointers: https://github.com/huggingface/transformers/blob/main/examples/pytorch/language-modeling/run_mlm.py

FutureWarning,
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:310: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this warning

FutureWarning,

***** Running training *****

Num examples = 11895

Num Epochs = 1

Instantaneous batch size per device = 8

Total train batch size (w. parallel, distributed & accumulation) = 8

Gradient Accumulation steps = 1

Total optimization steps = 1487

Number of trainable parameters = 124439808

[1487/1487 07:56, Epoch 1/1]

Step	Training Loss
------	---------------

500	3.887500
-----	----------

1000	3.618800
------	----------

Saving model checkpoint to /content/result/checkpoint-500

Configuration saved in /content/result/checkpoint-500/config.json

Model weights saved in /content/result/checkpoint-500/pytorch_model.bin

Saving model checkpoint to /content/result/checkpoint-1000

Configuration saved in /content/result/checkpoint-1000/config.json

Model weights saved in /content/result/checkpoint-1000/pytorch_model.bin

Training completed. Do not forget to share your model on huggingface.co/models =)

Saving model checkpoint to /content/result

Configuration saved in /content/result/config.json

Model weights saved in /content/result/pytorch_model.bin

Task 5: Creating a Tweet Generator

Now that we have our trained model, it's time to generate some Tweets. Since we

should have saved our model and tokenizer to an output directory I've already

should have saved our model and tokenizer to an output directory, I've already made some helper functions to load those in. We will focus on our *generate_text* function. The function will take as input some start text like "I am" or "My country is", etc., as well as a *max_length* parameter which tells the model how much text to generate. Let's Go!

In [133...

```
def load_model(model_path):
    model = GPT2LMHeadModel.from_pretrained(model_path)
    return model

def load_tokenizer(tokenizer_path):
    tokenizer = GPT2Tokenizer.from_pretrained(tokenizer_path)
    return tokenizer

def generate_text(sequence, max_length):
    #TODO: Load in the finetuned model and tokenizer
    model = load_model('/content/result')
    tokenizer = load_tokenizer('gpt2')

    # Encode our passed sequence
    ids = tokenizer.encode(
        #TODO: Put something here,
        text = sequence,
        max_length = max_length,
        return_tensors='pt'
    )
    #TODO: Use the generate method in our model class to generate output to
    final_outputs = model.generate(
        #TODO: Complete some of the parameters and leave the parameters I've
        ids,
        max_new_tokens = max_length,
        # Parameters you should leave
        do_sample = True,
        pad_token_id = model.config.eos_token_id,
        top_k = 50,
        top_p = 0.95,
    )
    # Function to print and decode output.
    print(tokenizer.decode(final_outputs[0], skip_special_tokens=True))
```

In [134...

```
#TODO: Now generate some text!
generate_text('I can make', 30)
```

loading configuration file /content/result/config.json