```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Creating a Sentiment Analysis Web App\n",
    "## Using PyTorch and SageMaker\n",
    "\n",
    "_Deep Learning Nanodegree Program | Deployment_\n",
    "\n",
    "---\n",
    "\n",
    "Now that we have a basic understanding of how SageMaker works we will try to use it to construct a complete project from end to end. Our goal will be to have a simple web page which a user can use to enter a movie review. The web page will then send the review off to our deployed model which will predict the sentiment of the entered review.\n",
    "\n",
    "## Instructions\n",
    "\n",
    "Some template code has already been provided for you, and you will need to implement additional functionality to successfully complete this notebook. You will not need to modify the included code beyond what is requested. Sections that begin with '**TODO**' in the header indicate that you need to complete or implement some portion within them. Instructions will be provided for each section and the specifics of the implementation are marked in the code block with a `# TODO: ...` comment. Please be sure to read the instructions carefully!\n",
    "\n",
    "In addition to implementing code, there will be questions for you to answer which relate to the task and your implementation. Each section where you will answer a question is preceded by a '**Question:**' header. Carefully read each question and provide your answer below the '**Answer:**' header by editing the Markdown cell.\n",
    "\n",
    "> **Note**: Code and Markdown cells can be executed using the **Shift+Enter** keyboard shortcut. In addition, a cell can be edited by typically clicking it (double-click for Markdown cells) or by pressing **Enter** while it is highlighted.\n",
    "\n",
    "## General Outline\n",
    "\n",
    "Recall the general outline for SageMaker projects using a notebook instance.\n",
    "\n",
    "1. Download or otherwise retrieve the data.\n",
```

```
    "2. Process / Prepare the data.\n",
    "3. Upload the processed data to S3.\n",
    "4. Train a chosen model.\n",
    "5. Test the trained model (typically using a batch transform
job).\n",
    "6. Deploy the trained model.\n",
    "7. Use the deployed model.\n",
    "\n",
    "For this project, you will be following the steps in the general
outline with some modifications. \n",
    "\n",
    "First, you will not be testing the model in its own step. You
will still be testing the model, however, you will do it by deploying
your model and then using the deployed model by sending the test data
to it. One of the reasons for doing this is so that you can make sure
that your deployed model is working correctly before moving forward.
\n",
    "\n",
    "In addition, you will deploy and use your trained model a second
time. In the second iteration you will customize the way that your
trained model is deployed by including some of your own code. In
addition, your newly deployed model will be used in the sentiment
analysis web app."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 1: Downloading the data\n",
    "\n",
    "As in the XGBoost in SageMaker notebook, we will be using the
[IMDb dataset](http://ai.stanford.edu/~amaas/data/sentiment/)\n",
    "\n",
    "> Maas, Andrew L., et al. [Learning Word Vectors for Sentiment
Analysis](http://ai.stanford.edu/~amaas/data/sentiment/). In
_Proceedings of the 49th Annual Meeting of the Association for
Computational Linguistics: Human Language Technologies_. Association
for Computational Linguistics, 2011."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
```

```
      "mkdir: cannot create directory ‘../data’: File exists\n",
      "--2020-05-06 13:35:22--  http://ai.stanford.edu/~amaas/data/
sentiment/aclImdb_v1.tar.gz\n",
      "Resolving ai.stanford.edu (ai.stanford.edu)... 171.64.68.10\n",
      "Connecting to ai.stanford.edu (ai.stanford.edu)|171.64.68.10|:
80... connected.\n",
      "HTTP request sent, awaiting response... 200 OK\n",
      "Length: 84125825 (80M) [application/x-gzip]\n",
      "Saving to: ‘../data/aclImdb_v1.tar.gz’\n",
      "\n",
      "../data/aclImdb_v1. 100%[===================>]  80.23M  5.70MB/
s    in 18s     \n",
      "\n",
      "2020-05-06 13:35:41 (4.35 MB/s) - ‘../data/aclImdb_v1.tar.gz’
saved [84125825/84125825]\n",
      "\n"
     ]
    }
   ],
   "source": [
    "%mkdir ../data\n",
    "!wget -O ../data/aclImdb_v1.tar.gz http://ai.stanford.edu/~amaas/
data/sentiment/aclImdb_v1.tar.gz\n",
    "!tar -zxf ../data/aclImdb_v1.tar.gz -C ../data"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 2: Preparing and Processing the data\n",
    "\n",
    "Also, as in the XGBoost notebook, we will be doing some initial
data processing. The first few steps are the same as in the XGBoost
example. To begin with, we will read in each of the reviews and
combine them into a single input structure. Then, we will split the
dataset into a training set and a testing set."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {},
   "outputs": [],
   "source": [
    "import os\n",
    "import glob\n",
    "\n",
    "def read_imdb_data(data_dir='../data/aclImdb'):\n",
    "    data = {}\n",
```

```
    "    labels = {}\n",
    "    \n",
    "    for data_type in ['train', 'test']:\n",
    "        data[data_type] = {}\n",
    "        labels[data_type] = {}\n",
    "        \n",
    "        for sentiment in ['pos', 'neg']:\n",
    "            data[data_type][sentiment] = []\n",
    "            labels[data_type][sentiment] = []\n",
    "            \n",
    "            path = os.path.join(data_dir, data_type, sentiment,
'*.txt')\n",
    "            files = glob.glob(path)\n",
    "            \n",
    "            for f in files:\n",
    "                with open(f) as review:\n",
    "                    data[data_type]
[sentiment].append(review.read())\n",
    "                    # Here we represent a positive review by '1'
and a negative review by '0'\n",
    "                    labels[data_type][sentiment].append(1 if
sentiment == 'pos' else 0)\n",
    "                    \n",
    "            assert len(data[data_type][sentiment]) ==
len(labels[data_type][sentiment]), \\\n",
    "                    \"{}/{} data size does not match labels
size\".format(data_type, sentiment)\n",
    "                \n",
    "    return data, labels"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "IMDB reviews: train = 12500 pos / 12500 neg, test = 12500 pos /
12500 neg\n"
     ]
    }
   ],
   "source": [
    "data, labels = read_imdb_data()\n",
    "print(\"IMDB reviews: train = {} pos / {} neg, test = {} pos / {}
neg\".format(\n",
    "            len(data['train']['pos']), len(data['train']['neg']),
```

```
\n",
    "            len(data['test']['pos']), len(data['test']['neg'])))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Now that we've read the raw training and testing data from the
downloaded dataset, we will combine the positive and negative reviews
and shuffle the resulting records."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sklearn.utils import shuffle\n",
    "\n",
    "def prepare_imdb_data(data, labels):\n",
    "    \"\"\"Prepare training and test sets from IMDb movie reviews.
\"\"\"\n",
    "    \n",
    "    #Combine positive and negative reviews and labels\n",
    "    data_train = data['train']['pos'] + data['train']['neg']\n",
    "    data_test = data['test']['pos'] + data['test']['neg']\n",
    "    labels_train = labels['train']['pos'] + labels['train']
['neg']\n",
    "    labels_test = labels['test']['pos'] + labels['test']['neg']
\n",
    "    \n",
    "    #Shuffle reviews and corresponding labels within training and
test sets\n",
    "    data_train, labels_train = shuffle(data_train, labels_train)
\n",
    "    data_test, labels_test = shuffle(data_test, labels_test)\n",
    "    \n",
    "    # Return a unified training data, test data, training labels,
test labets\n",
    "    return data_train, data_test, labels_train, labels_test"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "metadata": {},
   "outputs": [
    {
```

```
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "IMDb reviews (combined): train = 25000, test = 25000\n"
     ]
    }
   ],
   "source": [
    "train_X, test_X, train_y, test_y = prepare_imdb_data(data,
labels)\n",
    "print(\"IMDb reviews (combined): train = {}, test = {}
\".format(len(train_X), len(test_X)))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Now that we have our training and testing sets unified and
prepared, we should do a quick check and see an example of the data
our model will be trained on. This is generally a good idea as it
allows you to see how each of the further processing steps affects the
reviews and it also ensures that the data has been loaded correctly."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Lesbian vampire film about a couple on holiday who are staying
on the grounds of what they think is an empty manor house but is
really being used as a pair of lesbian vampires. As the vampires bring
in the occasional victim the couple go about their business until the
two groups come crashing together.<br /><br />Great looking film with
two very sexy women as the vampires there is nothing beyond the eye
candy that they provide to recommend this cult film. Yes its a sexy
vampire story. No it is not remotely interesting beyond the women. To
be honest there is a reason that I've been seeing stills of this film
in horror books and magazines it looks great, but other than
that...<br /><br />For those who want to see sexy vampires only.\n",
      "0\n"
     ]
    }
   ],
   "source": [
```

```
    "print(train_X[100])\n",
    "print(train_y[100])"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The first step in processing the reviews is to make sure that any
html tags that appear should be removed. In addition we wish to
tokenize our input, that way words such as *entertained* and
*entertaining* are considered the same with regard to sentiment
analysis."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 7,
   "metadata": {},
   "outputs": [],
   "source": [
    "import nltk\n",
    "from nltk.corpus import stopwords\n",
    "from nltk.stem.porter import *\n",
    "\n",
    "import re\n",
    "from bs4 import BeautifulSoup\n",
    "\n",
    "def review_to_words(review):\n",
    "    nltk.download(\"stopwords\", quiet=True)\n",
    "    stemmer = PorterStemmer()\n",
    "    \n",
    "    text = BeautifulSoup(review, \"html.parser\").get_text() #
Remove HTML tags\n",
    "    text = re.sub(r\"[^a-zA-Z0-9]\", \" \", text.lower()) #
Convert to lower case\n",
    "    words = text.split() # Split string into words\n",
    "    words = [w for w in words if w not in
stopwords.words(\"english\")] # Remove stopwords\n",
    "    words = [PorterStemmer().stem(w) for w in words] # stem\n",
    "    \n",
    "    return words"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The `review_to_words` method defined above uses `BeautifulSoup`
to remove any html tags that appear and uses the `nltk` package to
```

```
tokenize the reviews. As a check to ensure we know how everything is working, try applying `review_to_words` to one of the reviews in the training set."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 8,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "['lesbian', 'vampir', 'film', 'coupl', 'holiday', 'stay', 'ground', 'think', 'empti', 'manor', 'hous', 'realli', 'use', 'pair', 'lesbian', 'vampir', 'vampir', 'bring', 'occasion', 'victim', 'coupl', 'go', 'busi', 'two', 'group', 'come', 'crash', 'togeth', 'great', 'look', 'film', 'two', 'sexi', 'women', 'vampir', 'noth', 'beyond', 'eye', 'candi', 'provid', 'recommend', 'cult', 'film', 'ye', 'sexi', 'vampir', 'stori', 'remot', 'interest', 'beyond', 'women', 'honest', 'reason', 'see', 'still', 'film', 'horror', 'book', 'magazin', 'look', 'great', 'want', 'see', 'sexi', 'vampir']\n"
     ]
    }
   ],
   "source": [
    "# TODO: Apply review_to_words to a review (train_X[100] or any other review)\n",
    "print(review_to_words(train_X[100]))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "**Question:** Above we mentioned that `review_to_words` method removes html formatting and allows us to tokenize the words found in a review, for example, converting *entertained* and *entertaining* into *entertain* so that they are treated as though they are the same word. What else, if anything, does this method do to the input?"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "**Answer:** This method converts everything to lowercase, splits string into words and removes stopwords.\n",
    "\n"
```

```
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The method below applies the `review_to_words` method to each of
the reviews in the training and testing datasets. In addition it
caches the results. This is because performing this processing step
can take a long time. This way if you are unable to complete the
notebook in the current session, you can come back without needing to
process the data a second time."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 9,
   "metadata": {},
   "outputs": [],
   "source": [
    "import pickle\n",
    "\n",
    "cache_dir = os.path.join(\"../cache\", \"sentiment_analysis\")  #
where to store cache files\n",
    "os.makedirs(cache_dir, exist_ok=True)  # ensure cache directory
exists\n",
    "\n",
    "def preprocess_data(data_train, data_test, labels_train,
labels_test,\n",
    "                    cache_dir=cache_dir,
cache_file=\"preprocessed_data.pkl\"):\n",
    "    \"\"\"Convert each review to words; read from cache if
available.\"\"\"\n",
    "\n",
    "    # If cache_file is not None, try to read from it first\n",
    "    cache_data = None\n",
    "    if cache_file is not None:\n",
    "        try:\n",
    "            with open(os.path.join(cache_dir, cache_file),
\"rb\") as f:\n",
    "                cache_data = pickle.load(f)\n",
    "            print(\"Read preprocessed data from cache file:\",
cache_file)\n",
    "        except:\n",
    "            pass  # unable to read from cache, but that's
okay\n",
    "    \n",
    "    # If cache is missing, then do the heavy lifting\n",
    "    if cache_data is None:\n",
    "        # Preprocess training and test data to obtain words for
```

```
each review\n",
    "        #words_train = list(map(review_to_words, data_train))\n",
    "        #words_test = list(map(review_to_words, data_test))\n",
    "        words_train = [review_to_words(review) for review in
data_train]\n",
    "        words_test = [review_to_words(review) for review in
data_test]\n",
    "        \n",
    "        # Write to cache file for future runs\n",
    "        if cache_file is not None:\n",
    "            cache_data = dict(words_train=words_train,
words_test=words_test,\n",
    "                              labels_train=labels_train,
labels_test=labels_test)\n",
    "            with open(os.path.join(cache_dir, cache_file),
\"wb\") as f:\n",
    "                pickle.dump(cache_data, f)\n",
    "            print(\"Wrote preprocessed data to cache file:\",
cache_file)\n",
    "    else:\n",
    "        # Unpack data loaded from cache file\n",
    "        words_train, words_test, labels_train, labels_test =
(cache_data['words_train'],\n",
    "                cache_data['words_test'],
cache_data['labels_train'], cache_data['labels_test'])\n",
    "    \n",
    "    return words_train, words_test, labels_train, labels_test"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 10,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Read preprocessed data from cache file:
preprocessed_data.pkl\n"
     ]
    }
   ],
   "source": [
    "# Preprocess data\n",
    "train_X, test_X, train_y, test_y = preprocess_data(train_X,
test_X, train_y, test_y)"
   ]
  },
  {
```

```
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Transform the data\n",
    "\n",
    "In the XGBoost notebook we transformed the data from its word representation to a bag-of-words feature representation. For the model we are going to construct in this notebook we will construct a feature representation which is very similar. To start, we will represent each word as an integer. Of course, some of the words that appear in the reviews occur very infrequently and so likely don't contain much information for the purposes of sentiment analysis. The way we will deal with this problem is that we will fix the size of our working vocabulary and we will only include the words that appear most frequently. We will then combine all of the infrequent words into a single category and, in our case, we will label it as `1`.\n",
    "\n",
    "Since we will be using a recurrent neural network, it will be convenient if the length of each review is the same. To do this, we will fix a size for our reviews and then pad short reviews with the category 'no word' (which we will label `0`) and truncate long reviews."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### (TODO) Create a word dictionary\n",
    "\n",
    "To begin with, we need to construct a way to map words that appear in the reviews to integers. Here we fix the size of our vocabulary (including the 'no word' and 'infrequent' categories) to be `5000` but you may wish to change this to see how it affects the model.\n",
    "\n",
    "> **TODO:** Complete the implementation for the `build_dict()` method below. Note that even though the vocab_size is set to `5000`, we only want to construct a mapping for the most frequently appearing `4998` words. This is because we want to reserve the special labels `0` for 'no word' and `1` for 'infrequent word'."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 11,
   "metadata": {},
   "outputs": [],
   "source": [
    "import numpy as np\n",
```

```
    "\n",
    "def build_dict(data, vocab_size = 5000):\n",
    "    \"\"\"Construct and return a dictionary mapping each of the
most frequently appearing words to a unique integer.\"\"\"\n",
    "    \n",
    "    # TODO: Determine how often each word appears in `data`. Note
that `data` is a list of sentences and that a\n",
    "    #       sentence is a list of words.\n",
    "    \n",
    "    \n",
    "    word_count = {} # A dict storing the words that appear in the
reviews along with how often they occur\n",
    "    \n",
    "    for sent in data:\n",
    "        for word in sent:\n",
    "            if word in word_count:\n",
    "                word_count[word]+=1\n",
    "            else:\n",
    "                word_count[word]=1\n",
    "    \n",
    "    # TODO: Sort the words found in `data` so that
sorted_words[0] is the most frequently appearing word and\n",
    "    #       sorted_words[-1] is the least frequently appearing
word.\n",
    "    \n",
    "    sorted_words = [item[0] for item in
sorted(word_count.items(), key = lambda x:x[1], reverse=True)]\n",
    "    \n",
    "    word_dict = {} # This is what we are building, a dictionary
that translates words into integers\n",
    "    for idx, word in enumerate(sorted_words[:vocab_size - 2]): #
The -2 is so that we save room for the 'no word'\n",
    "        word_dict[word] = idx + 2                             #
'infrequent' labels\n",
    "        \n",
    "    return word_dict"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 12,
   "metadata": {},
   "outputs": [],
   "source": [
    "word_dict = build_dict(train_X)"
   ]
  },
  {
   "cell_type": "markdown",
```

```
   "metadata": {},
   "source": [
    "**Question:** What are the five most frequently appearing
(tokenized) words in the training set? Does it makes sense that these
words appear frequently in the training set?"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "**Answer:** The 5 most frequently appearing words in the training
set are : movi, film, one, like and time. Yes these words appear
frequently in the training set."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 13,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "movi\n",
      "film\n",
      "one\n",
      "like\n",
      "time\n"
     ]
    }
   ],
   "source": [
    "# TODO: Use this space to determine the five most frequently
appearing words in the training set.\n",
    "idx=0\n",
    "for word in word_dict:\n",
    "    print(word)\n",
    "    idx+=1\n",
    "    if idx==5:\n",
    "        break"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Save `word_dict`\n",
    "\n",
```

```
    "Later on when we construct an endpoint which processes a
submitted review we will need to make use of the `word_dict` which we
have created. As such, we will save it to a file now for future use."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 14,
   "metadata": {},
   "outputs": [],
   "source": [
    "data_dir = '../data/pytorch' # The folder we will use for storing
data\n",
    "if not os.path.exists(data_dir): # Make sure that the folder
exists\n",
    "    os.makedirs(data_dir)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 15,
   "metadata": {},
   "outputs": [],
   "source": [
    "with open(os.path.join(data_dir, 'word_dict.pkl'), \"wb\") as f:
\n",
    "    pickle.dump(word_dict, f)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Transform the reviews\n",
    "\n",
    "Now that we have our word dictionary which allows us to transform
the words appearing in the reviews into integers, it is time to make
use of it and convert our reviews to their integer sequence
representation, making sure to pad or truncate to a fixed length,
which in our case is `500`."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 16,
   "metadata": {},
   "outputs": [],
   "source": [
    "def convert_and_pad(word_dict, sentence, pad=500):\n",
    "    NOWORD = 0 # We will use 0 to represent the 'no word'
```

```
category\n",
    "    INFREQ = 1 # and we use 1 to represent the infrequent words,
i.e., words not appearing in word_dict\n",
    "    \n",
    "    working_sentence = [NOWORD] * pad\n",
    "    \n",
    "    for word_index, word in enumerate(sentence[:pad]):\n",
    "        if word in word_dict:\n",
    "            working_sentence[word_index] = word_dict[word]\n",
    "        else:\n",
    "            working_sentence[word_index] = INFREQ\n",
    "            \n",
    "    return working_sentence, min(len(sentence), pad)\n",
    "\n",
    "def convert_and_pad_data(word_dict, data, pad=500):\n",
    "    result = []\n",
    "    lengths = []\n",
    "    \n",
    "    for sentence in data:\n",
    "        converted, leng = convert_and_pad(word_dict, sentence,
pad)\n",
    "        result.append(converted)\n",
    "        lengths.append(leng)\n",
    "        \n",
    "    return np.array(result), np.array(lengths)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 17,
   "metadata": {},
   "outputs": [],
   "source": [
    "train_X, train_X_len = convert_and_pad_data(word_dict, train_X)
\n",
    "test_X, test_X_len = convert_and_pad_data(word_dict, test_X)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "As a quick check to make sure that things are working as
intended, check to see what one of the reviews in the training set
looks like after having been processeed. Does this look reasonable?
What is the length of a review in the training set?"
   ]
  },
  {
   "cell_type": "code",
```

   "execution_count": 18,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "array([2, 146, 1939,   1, 110,   90,   1, 207,   8,
2,   26,\n",
       "   301,   71,   43,   1, 2842, 511,   37, 3486, 762,
103,   28,\n",
       "   298, 1377,   2, 664, 552, 197, 198, 451, 284,
2,   46,\n",
       "   3846, 117, 1510, 1022,   38,   37, 430, 282, 3816,
781,   29,\n",
       "   1696, 1647,   9, 248,   53, 379,   43, 788,   1,
553,   51,\n",
       "   1470, 1363,   3, 1081, 3220, 602, 2275, 680,   24,
445, 190,\n",
       "   1501, 337, 769,   38, 226, 636, 147, 1501, 337,
30,   2,\n",
       "   2424, 565,   4, 212,   1,   1, 535, 517,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",
       "   0,   0,   0,   0,   0,   0,   0,   0,   0,
0,   0,\n",

```
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0,     0,     0,     0,     0,
0,     0,\n",
"         0,     0,     0,     0,     0])"
]
},
"execution_count": 18,
```

```
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "# Use this cell to examine one of the processed reviews to make sure everything is working as intended.\n",
    "train_X[15]"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "**Question:** In the cells above we use the `preprocess_data` and `convert_and_pad_data` methods to process both the training and testing set. Why or why not might this be a problem?"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "**Answer:** 1) preprocess_data helps to cut time and cost in processing process as data loaded to disk(cached) is easily loaded back when needed\n",
    "\n",
    "2) convert_and_pad_data cuts the reviews if length exceeds pad length which might remove important information and hide certain sentiments affecting the model performance"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 3: Upload the data to S3\n",
    "\n",
    "As in the XGBoost notebook, we will need to upload the training dataset to S3 in order for our training code to access it. For now we will save it locally and we will upload to S3 later on.\n",
    "\n",
    "### Save the processed training dataset locally\n",
    "\n",
    "It is important to note the format of the data that we are saving as we will need to know it when we write the training code. In our case, each row of the dataset has the form `label`, `length`, `review[500]` where `review[500]` is a sequence of `500` integers representing the words in the review."
   ]
```

```
  },
  {
   "cell_type": "code",
   "execution_count": 19,
   "metadata": {},
   "outputs": [],
   "source": [
    "import pandas as pd\n",
    "    \n",
    "pd.concat([pd.DataFrame(train_y), pd.DataFrame(train_X_len), 
pd.DataFrame(train_X)], axis=1) \\\n",
    "        .to_csv(os.path.join(data_dir, 'train.csv'), 
header=False, index=False)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Uploading the training data\n",
    "\n",
    "\n",
    "Next, we need to upload the training data to the SageMaker 
default S3 bucket so that we can provide access to it while training 
our model."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 20,
   "metadata": {},
   "outputs": [],
   "source": [
    "import sagemaker\n",
    "\n",
    "sagemaker_session = sagemaker.Session()\n",
    "\n",
    "bucket = sagemaker_session.default_bucket()\n",
    "prefix = 'sagemaker/sentiment_rnn'\n",
    "\n",
    "role = sagemaker.get_execution_role()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 21,
   "metadata": {},
   "outputs": [],
   "source": [
    "input_data = sagemaker_session.upload_data(path=data_dir,
```

```
bucket=bucket, key_prefix=prefix)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "**NOTE:** The cell above uploads the entire contents of our data directory. This includes the `word_dict.pkl` file. This is fortunate as we will need this later on when we create an endpoint that accepts an arbitrary review. For now, we will just take note of the fact that it resides in the data directory (and so also in the S3 training bucket) and that we will need to make sure it gets saved in the model directory."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 4: Build and Train the PyTorch Model\n",
    "\n",
    "In the XGBoost notebook we discussed what a model is in the SageMaker framework. In particular, a model comprises three objects\n",
    "\n",
    " - Model Artifacts,\n",
    " - Training Code, and\n",
    " - Inference Code,\n",
    " \n",
    "each of which interact with one another. In the XGBoost example we used training and inference code that was provided by Amazon. Here we will still be using containers provided by Amazon with the added benefit of being able to include our own custom code.\n",
    "\n",
    "We will start by implementing our own neural network in PyTorch along with a training script. For the purposes of this project we have provided the necessary model object in the `model.py` file, inside of the `train` folder. You can see the provided implementation by running the cell below."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 22,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
```

```
     "text": [
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mtorch.nn\u001b[39;49;00m
\u001b[34mas\u001b[39;49;00m
\u001b[04m\u001b[36mnn\u001b[39;49;00m\r\n",
      "\r\n",
      "\u001b[34mclass\u001b[39;49;00m
\u001b[04m\u001b[32mLSTMClassifier\u001b[39;49;00m(nn.Module):\r\n",
      "    \u001b[33m\"\"\"\u001b[39;49;00m\r\n",
      "\u001b[33m    This is the simple RNN model we will be using to
perform Sentiment Analysis.\u001b[39;49;00m\r\n",
      "\u001b[33m    \"\"\"\u001b[39;49;00m\r\n",
      "\r\n",
      "    \u001b[34mdef\u001b[39;49;00m
\u001b[32m__init__\u001b[39;49;00m(\u001b[36mself\u001b[39;49;00m,
embedding_dim, hidden_dim, vocab_size):\r\n",
      "        \u001b[33m\"\"\"\u001b[39;49;00m\r\n",
      "\u001b[33m        Initialize the model by settingg up the
various layers.\u001b[39;49;00m\r\n",
      "\u001b[33m        \"\"\"\u001b[39;49;00m\r\n",
      "        \u001b[36msuper\u001b[39;49;00m(LSTMClassifier,
\u001b[36mself\u001b[39;49;00m).\u001b[32m__init__\u001b[39;49;00m()
\r\n",
      "\r\n",
      "        \u001b[36mself\u001b[39;49;00m.embedding =
nn.Embedding(vocab_size, embedding_dim,
padding_idx=\u001b[34m0\u001b[39;49;00m)\r\n",
      "        \u001b[36mself\u001b[39;49;00m.lstm =
nn.LSTM(embedding_dim, hidden_dim)\r\n",
      "        \u001b[36mself\u001b[39;49;00m.dense =
nn.Linear(in_features=hidden_dim,
out_features=\u001b[34m1\u001b[39;49;00m)\r\n",
      "        \u001b[36mself\u001b[39;49;00m.sig = nn.Sigmoid()\r\n",
      "        \r\n",
      "        \u001b[36mself\u001b[39;49;00m.word_dict =
\u001b[36mNone\u001b[39;49;00m\r\n",
      "\r\n",
      "    \u001b[34mdef\u001b[39;49;00m
\u001b[32mforward\u001b[39;49;00m(\u001b[36mself\u001b[39;49;00m, x):
\r\n",
      "        \u001b[33m\"\"\"\u001b[39;49;00m\r\n",
      "\u001b[33m        Perform a forward pass of our model on some
input.\u001b[39;49;00m\r\n",
      "\u001b[33m        \"\"\"\u001b[39;49;00m\r\n",
      "        x = x.t()\r\n",
      "        lengths = x[\u001b[34m0\u001b[39;49;00m,:]\r\n",
      "        reviews = x[\u001b[34m1\u001b[39;49;00m:,:]\r\n",
      "        embeds =
\u001b[36mself\u001b[39;49;00m.embedding(reviews)\r\n",
      "        lstm_out, _ =
```

```
\u001b[36mself\u001b[39;49;00m.lstm(embeds)\r\n",
      "        out = \u001b[36mself\u001b[39;49;00m.dense(lstm_out)
\r\n",
      "        out = out[lengths - \u001b[34m1\u001b[39;49;00m,
\u001b[36mrange\u001b[39;49;00m(\u001b[36mlen\u001b[39;49;00m(lengths)
)]\r\n",
      "        \u001b[34mreturn\u001b[39;49;00m
\u001b[36mself\u001b[39;49;00m.sig(out.squeeze())\r\n"
     ]
    }
   ],
   "source": [
    "!pygmentize train/model.py"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The important takeaway from the implementation provided is that
there are three parameters that we may wish to tweak to improve the
performance of our model. These are the embedding dimension, the
hidden dimension and the size of the vocabulary. We will likely want
to make these parameters configurable in the training script so that
if we wish to modify them we do not need to modify the script itself.
We will see how to do this later on. To start we will write some of
the training code in the notebook so that we can more easily diagnose
any issues that arise.\n",
    "\n",
    "First we will load a small portion of the training data set to
use as a sample. It would be very time consuming to try and train the
model completely in the notebook as we do not have access to a gpu and
the compute instance that we are using is not particularly powerful.
However, we can work on a small bit of the data to get a feel for how
our training script is behaving."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 23,
   "metadata": {},
   "outputs": [],
   "source": [
    "import torch\n",
    "import torch.utils.data\n",
    "\n",
    "# Read in only the first 250 rows\n",
    "train_sample = pd.read_csv(os.path.join(data_dir, 'train.csv'),
header=None, names=None, nrows=250)\n",
    "\n",
```

```
    "# Turn the input pandas dataframe into tensors\n",
    "train_sample_y =
torch.from_numpy(train_sample[[0]].values).float().squeeze()\n",
    "train_sample_X = torch.from_numpy(train_sample.drop([0],
axis=1).values).long()\n",
    "\n",
    "# Build the dataset\n",
    "train_sample_ds = torch.utils.data.TensorDataset(train_sample_X,
train_sample_y)\n",
    "# Build the dataloader\n",
    "train_sample_dl = torch.utils.data.DataLoader(train_sample_ds,
batch_size=50)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### (TODO) Writing the training method\n",
    "\n",
    "Next we need to write the training code itself. This should be
very similar to training methods that you have written before to train
PyTorch models. We will leave any difficult aspects such as model
saving / loading and parameter loading until a little later."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 24,
   "metadata": {},
   "outputs": [],
   "source": [
    "def train(model, train_loader, epochs, optimizer, loss_fn,
device):\n",
    "    for epoch in range(1, epochs + 1):\n",
    "        model.train()\n",
    "        total_loss = 0\n",
    "        for batch in train_loader:         \n",
    "            batch_X, batch_y = batch\n",
    "            \n",
    "            batch_X = batch_X.to(device)\n",
    "            batch_y = batch_y.to(device)\n",
    "            \n",
    "            # TODO: Complete this train method to train the model
provided.\n",
    "            optimizer.zero_grad()\n",
    "            \n",
    "            output = model.forward(batch_X)\n",
    "            \n",
    "            loss = loss_fn(output, batch_y)\n",
```

```
    "            \n",
    "            loss.backward()\n",
    "            \n",
    "            optimizer.step()\n",
    "            \n",
    "            total_loss += loss.data.item()\n",
    "            \n",
    "        print(\"Epoch: {}, BCELoss: {}\".format(epoch,
total_loss / len(train_loader)))"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Supposing we have the training method above, we will test that it
is working by writing a bit of code in the notebook that executes our
training method on the small sample training set that we loaded
earlier. The reason for doing this in the notebook is so that we have
an opportunity to fix any errors that arise early when they are easier
to diagnose."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 25,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Epoch: 1, BCELoss: 0.6955794095993042\n",
      "Epoch: 2, BCELoss: 0.6861237287521362\n",
      "Epoch: 3, BCELoss: 0.6778116345405578\n",
      "Epoch: 4, BCELoss: 0.6681714057922363\n",
      "Epoch: 5, BCELoss: 0.6552742123603821\n"
     ]
    }
   ],
   "source": [
    "import torch.optim as optim\n",
    "from train.model import LSTMClassifier\n",
    "\n",
    "device = torch.device(\"cuda\" if torch.cuda.is_available() else
\"cpu\")\n",
    "model = LSTMClassifier(32, 100, 5000).to(device)\n",
    "optimizer = optim.Adam(model.parameters())\n",
    "loss_fn = torch.nn.BCELoss()\n",
    "\n",
```

```
    "train(model, train_sample_dl, 5, optimizer, loss_fn, device)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "In order to construct a PyTorch model using SageMaker we must provide SageMaker with a training script. We may optionally include a directory which will be copied to the container and from which our training code will be run. When the training container is executed it will check the uploaded directory (if there is one) for a `requirements.txt` file and install any required Python libraries, after which the training script will be run."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### (TODO) Training the model\n",
    "\n",
    "When a PyTorch model is constructed in SageMaker, an entry point must be specified. This is the Python file which will be executed when the model is trained. Inside of the `train` directory is a file called `train.py` which has been provided and which contains most of the necessary code to train our model. The only thing that is missing is the implementation of the `train()` method which you wrote earlier in this notebook.\n",
    "\n",
    "**TODO**: Copy the `train()` method written above and paste it into the `train/train.py` file where required.\n",
    "\n",
    "The way that SageMaker passes hyperparameters to the training script is by way of arguments. These arguments can then be parsed and used in the training script. To see how this is done take a look at the provided `train/train.py` file."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 26,
   "metadata": {},
   "outputs": [],
   "source": [
    "from sagemaker.pytorch import PyTorch\n",
    "\n",
    "estimator = PyTorch(entry_point=\"train.py\",\n",
    "                    source_dir=\"train\",\n",
    "                    role=role,\n",
```

```
    "                    framework_version='0.4.0',\n",
    "                    train_instance_count=1,\n",
    "                    train_instance_type='ml.p2.xlarge',\n",
    "                    hyperparameters={\n",
    "                        'epochs': 10,\n",
    "                        'hidden_dim': 200,\n",
    "                    })"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 27,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "2020-05-06 13:38:10 Starting - Starting the training job...\n",
      "2020-05-06 13:38:12 Starting - Launching requested ML
instances......\n",
      "2020-05-06 13:39:37 Starting - Preparing the instances for
training......\n",
      "2020-05-06 13:40:30 Downloading - Downloading input data...\n",
      "2020-05-06 13:41:07 Training - Downloading the training
image...\n",
      "2020-05-06 13:41:34 Training - Training image download
completed. Training in progress..\u001b[34mbash: cannot set terminal
process group (-1): Inappropriate ioctl for device\u001b[0m\n",
      "\u001b[34mbash: no job control in this shell\u001b[0m\n",
      "\u001b[34m2020-05-06 13:41:35,278 sagemaker-containers INFO
Imported framework sagemaker_pytorch_container.training\u001b[0m\n",
      "\u001b[34m2020-05-06 13:41:35,301
sagemaker_pytorch_container.training INFO     Block until all host DNS
lookups succeed.\u001b[0m\n",
      "\u001b[34m2020-05-06 13:41:38,322
sagemaker_pytorch_container.training INFO     Invoking user training
script.\u001b[0m\n",
      "\u001b[34m2020-05-06 13:41:38,529 sagemaker-containers INFO
Module train does not provide a setup.py. \u001b[0m\n",
      "\u001b[34mGenerating setup.py\u001b[0m\n",
      "\u001b[34m2020-05-06 13:41:38,529 sagemaker-containers INFO
Generating setup.cfg\u001b[0m\n",
      "\u001b[34m2020-05-06 13:41:38,529 sagemaker-containers INFO
Generating MANIFEST.in\u001b[0m\n",
      "\u001b[34m2020-05-06 13:41:38,529 sagemaker-containers INFO
Installing module with the following command:\u001b[0m\n",
      "\u001b[34m/usr/bin/python -m pip install -U . -r
requirements.txt\u001b[0m\n",
      "\u001b[34mProcessing /opt/ml/code\u001b[0m\n",
```

```
      "\u001b[34mCollecting pandas (from -r requirements.txt (line 1))
\u001b[0m\n",
      "\u001b[34m  Downloading https://files.pythonhosted.org/
packages/
74/24/0cdbf8907e1e3bc5a8da03345c23cbed7044330bb8f73bb12e711a640a00/
pandas-0.24.2-cp35-cp35m-manylinux1_x86_64.whl (10.0MB)\u001b[0m\n",
      "\u001b[34mCollecting numpy (from -r requirements.txt (line 2))
\n",
      "  Downloading https://files.pythonhosted.org/packages/38/92/
fa5295d9755c7876cb8490eab866e1780154033fa45978d9cf74ffbd4c68/
numpy-1.18.4-cp35-cp35m-manylinux1_x86_64.whl (20.0MB)\u001b[0m\n",
      "\u001b[34mCollecting nltk (from -r requirements.txt (line 3))
\n",
      "  Downloading https://files.pythonhosted.org/packages/92/75/
ce35194d8e3022203cca0d2f896dbb88689f9b3fce8e9f9cff942913519d/
nltk-3.5.zip (1.4MB)\u001b[0m\n",
      "\u001b[34mCollecting beautifulsoup4 (from -r requirements.txt
(line 4))\n",
      "  Downloading https://files.pythonhosted.org/packages/e8/
b5/7bb03a696f2c9b7af792a8f51b82974e51c268f15e925fc834876a4efa0b/
beautifulsoup4-4.9.0-py3-none-any.whl (109kB)\u001b[0m\n",
      "\u001b[34mCollecting html5lib (from -r requirements.txt (line
5))\n",
      "  Downloading https://files.pythonhosted.org/packages/a5/62/
bbd2be0e7943ec8504b517e62bab011b4946e1258842bc159e5dfde15b96/
html5lib-1.0.1-py2.py3-none-any.whl (117kB)\u001b[0m\n",
      "\u001b[34mRequirement already satisfied, skipping upgrade:
python-dateutil>=2.5.0 in /usr/local/lib/python3.5/dist-packages (from
pandas->-r requirements.txt (line 1)) (2.7.5)\u001b[0m\n",
      "\u001b[34mCollecting pytz>=2011k (from pandas->-r
requirements.txt (line 1))\n",
      "  Downloading https://files.pythonhosted.org/packages/4f/
a4/879454d49688e2fad93e59d7d4efda580b783c745fd2ec2a3adf87b0808d/
pytz-2020.1-py2.py3-none-any.whl (510kB)\u001b[0m\n",
      "\u001b[34mRequirement already satisfied, skipping upgrade:
click in /usr/local/lib/python3.5/dist-packages (from nltk->-r
requirements.txt (line 3)) (7.0)\u001b[0m\n",
      "\u001b[34mCollecting joblib (from nltk->-r requirements.txt
(line 3))\n",
      "  Downloading https://files.pythonhosted.org/packages/28/5c/
cf6a2b65a321c4a209efcdf64c2689efae2cb62661f8f6f4bb28547cf1bf/
joblib-0.14.1-py2.py3-none-any.whl (294kB)\u001b[0m\n",
      "\u001b[34mCollecting regex (from nltk->-r requirements.txt
(line 3))\u001b[0m\n",
      "\u001b[34m  Downloading https://files.pythonhosted.org/
packages/4c/e7/
eee73c42c1193fecc0e91361a163cbb8dfbea62c3db7618ad986e5b43a14/
regex-2020.4.4.tar.gz (695kB)\u001b[0m\n",
      "\u001b[34mCollecting tqdm (from nltk->-r requirements.txt (line
3))\n",
```

```
      "  Downloading https://files.pythonhosted.org/packages/
c9/40/058b12e8ba10e35f89c9b1fdfc2d4c7f8c05947df2d5eb3c7b258019fda0/
tqdm-4.46.0-py2.py3-none-any.whl (63kB)\u001b[0m\n",
      "\u001b[34mCollecting soupsieve>1.2 (from beautifulsoup4->-r
requirements.txt (line 4))\u001b[0m\n",
      "\u001b[34m  Downloading https://files.pythonhosted.org/
packages/05/cf/
ea245e52f55823f19992447b008bcbb7f78efc5960d77f6c34b5b45b36dd/
soupsieve-2.0-py2.py3-none-any.whl\u001b[0m\n",
      "\u001b[34mRequirement already satisfied, skipping upgrade:
six>=1.9 in /usr/local/lib/python3.5/dist-packages (from html5lib->-r
requirements.txt (line 5)) (1.11.0)\u001b[0m\n",
      "\u001b[34mCollecting webencodings (from html5lib->-r
requirements.txt (line 5))\n",
      "  Downloading https://files.pythonhosted.org/packages/
f4/24/2a3e3df732393fed8b3ebf2ec078f05546de641fe1b667ee316ec1dcf3b7/
webencodings-0.5.1-py2.py3-none-any.whl\u001b[0m\n",
      "\u001b[34mBuilding wheels for collected packages: nltk, train,
regex\n",
      "  Running setup.py bdist_wheel for nltk: started\n",
      "  Running setup.py bdist_wheel for nltk: finished with status
'done'\n",
      "  Stored in directory: /root/.cache/pip/wheels/ae/8c/3f/
b1fe0ba04555b08b57ab52ab7f86023639a526d8bc8d384306\n",
      "  Running setup.py bdist_wheel for train: started\u001b[0m\n",
      "\u001b[34m  Running setup.py bdist_wheel for train: finished
with status 'done'\n",
      "  Stored in directory: /tmp/pip-ephem-wheel-cache-qdp34_be/
wheels/35/24/16/37574d11bf9bde50616c67372a334f94fa8356bc7164af8ca3\n",
      "  Running setup.py bdist_wheel for regex: started\u001b[0m\n",
      "\u001b[34m  Running setup.py bdist_wheel for regex: finished
with status 'done'\n",
      "  Stored in directory: /root/.cache/pip/wheels/e6/9b/ae/
2972da29cc7759b71dee015813b7c6931917d6a51e64ed5e79\u001b[0m\n",
      "\u001b[34mSuccessfully built nltk train regex\u001b[0m\n",
      "\u001b[34mInstalling collected packages: numpy, pytz, pandas,
joblib, regex, tqdm, nltk, soupsieve, beautifulsoup4, webencodings,
html5lib, train\n",
      "  Found existing installation: numpy 1.15.4\u001b[0m\n",
      "\u001b[34m    Uninstalling numpy-1.15.4:\n",
      "      Successfully uninstalled numpy-1.15.4\u001b[0m\n",
      "\u001b[34mSuccessfully installed beautifulsoup4-4.9.0
html5lib-1.0.1 joblib-0.14.1 nltk-3.5 numpy-1.18.4 pandas-0.24.2
pytz-2020.1 regex-2020.4.4 soupsieve-2.0 tqdm-4.46.0 train-1.0.0
webencodings-0.5.1\u001b[0m\n",
      "\u001b[34mYou are using pip version 18.1, however version 20.1
is available.\u001b[0m\n",
      "\u001b[34mYou should consider upgrading via the 'pip install --
upgrade pip' command.\u001b[0m\n",
      "\u001b[34m2020-05-06 13:42:01,643 sagemaker-containers INFO
```

```
Invoking user script\n",
      "\u001b[0m\n",
      "\u001b[34mTraining Env:\n",
      "\u001b[0m\n",
      "\u001b[34m{\n",
      "    \"num_cpus\": 4,\n",
      "    \"additional_framework_parameters\": {},\n",
      "    \"hyperparameters\": {\n",
      "        \"epochs\": 10,\n",
      "        \"hidden_dim\": 200\n",
      "    },\n",
      "    \"output_data_dir\": \"/opt/ml/output/data\",\n",
      "    \"output_intermediate_dir\": \"/opt/ml/output/
intermediate\",\n",
      "    \"output_dir\": \"/opt/ml/output\",\n",
      "    \"input_config_dir\": \"/opt/ml/input/config\",\n",
      "    \"channel_input_dirs\": {\n",
      "        \"training\": \"/opt/ml/input/data/training\"\n",
      "    },\n",
      "    \"current_host\": \"algo-1\",\n",
      "    \"framework_module\":
\"sagemaker_pytorch_container.training:main\",\n",
      "    \"job_name\": \"sagemaker-
pytorch-2020-05-06-13-38-10-077\",\n",
      "    \"user_entry_point\": \"train.py\",\n",
      "    \"hosts\": [\n",
      "        \"algo-1\"\n",
      "    ],\n",
      "    \"network_interface_name\": \"eth0\",\n",
      "    \"module_dir\": \"s3://sagemaker-ap-south-1-267156467824/
sagemaker-pytorch-2020-05-06-13-38-10-077/source/sourcedir.tar.gz\",
\n",
      "    \"model_dir\": \"/opt/ml/model\",\n",
      "    \"resource_config\": {\n",
      "        \"hosts\": [\n",
      "            \"algo-1\"\n",
      "        ],\n",
      "        \"network_interface_name\": \"eth0\",\n",
      "        \"current_host\": \"algo-1\"\n",
      "    },\n",
      "    \"module_name\": \"train\",\n",
      "    \"num_gpus\": 1,\n",
      "    \"input_data_config\": {\n",
      "        \"training\": {\n",
      "            \"S3DistributionType\": \"FullyReplicated\",\n",
      "            \"RecordWrapperType\": \"None\",\n",
      "            \"TrainingInputMode\": \"File\"\n",
      "        }\n",
      "    },\n",
      "    \"log_level\": 20,\n",
```

```
      "    \"input_dir\": \"/opt/ml/input\"\u001b[0m\n",
      "\u001b[34m}\n",
      "\u001b[0m\n",
      "\u001b[34mEnvironment variables:\n",
      "\u001b[0m\n",
      "\u001b[34mSM_OUTPUT_INTERMEDIATE_DIR=/opt/ml/output/
intermediate\u001b[0m\n",
      "\u001b[34mSM_CHANNEL_TRAINING=/opt/ml/input/data/
training\u001b[0m\n",
      "\u001b[34mSM_CHANNELS=[\"training\"]\u001b[0m\n",
      "\u001b[34mPYTHONPATH=/usr/local/bin:/usr/lib/python35.zip:/usr/
lib/python3.5:/usr/lib/python3.5/plat-x86_64-linux-gnu:/usr/lib/
python3.5/lib-dynload:/usr/local/lib/python3.5/dist-packages:/usr/lib/
python3/dist-packages\u001b[0m\n",
      "\u001b[34mSM_MODULE_DIR=s3://sagemaker-ap-south-1-267156467824/
sagemaker-pytorch-2020-05-06-13-38-10-077/source/
sourcedir.tar.gz\u001b[0m\n",
      "\u001b[34mSM_INPUT_CONFIG_DIR=/opt/ml/input/config\u001b[0m\n",
      "\u001b[34mSM_TRAINING_ENV={\"additional_framework_parameters\":
{},\"channel_input_dirs\":{\"training\":\"/opt/ml/input/data/
training\"},\"current_host\":\"algo-1\",\"framework_module\":
\"sagemaker_pytorch_container.training:main\",\"hosts\":[\"algo-1\"],
\"hyperparameters\":{\"epochs\":10,\"hidden_dim\":200},
\"input_config_dir\":\"/opt/ml/input/config\",\"input_data_config\":
{\"training\":{\"RecordWrapperType\":\"None\",\"S3DistributionType\":
\"FullyReplicated\",\"TrainingInputMode\":\"File\"}},\"input_dir\":\"/
opt/ml/input\",\"job_name\":\"sagemaker-
pytorch-2020-05-06-13-38-10-077\",\"log_level\":20,\"model_dir\":\"/
opt/ml/model\",\"module_dir\":\"s3://sagemaker-ap-
south-1-267156467824/sagemaker-pytorch-2020-05-06-13-38-10-077/source/
sourcedir.tar.gz\",\"module_name\":\"train\",
\"network_interface_name\":\"eth0\",\"num_cpus\":4,\"num_gpus\":
1,\"output_data_dir\":\"/opt/ml/output/data\",\"output_dir\":\"/opt/
ml/output\",\"output_intermediate_dir\":\"/opt/ml/output/
intermediate\",\"resource_config\":{\"current_host\":\"algo-1\",
\"hosts\":[\"algo-1\"],\"network_interface_name\":\"eth0\"},
\"user_entry_point\":\"train.py\"}\u001b[0m\n",
      "\u001b[34mSM_NETWORK_INTERFACE_NAME=eth0\u001b[0m\n",
      "\u001b[34mSM_FRAMEWORK_PARAMS={}\u001b[0m\n",
      "\u001b[34mSM_INPUT_DIR=/opt/ml/input\u001b[0m\n",
      "\u001b[34mSM_NUM_CPUS=4\u001b[0m\n",
      "\u001b[34mSM_MODULE_NAME=train\u001b[0m\n",
      "\u001b[34mSM_OUTPUT_DIR=/opt/ml/output\u001b[0m\n",
      "\u001b[34mSM_MODEL_DIR=/opt/ml/model\u001b[0m\n",

"\u001b[34mSM_FRAMEWORK_MODULE=sagemaker_pytorch_container.training:ma
in\u001b[0m\n",
      "\u001b[34mSM_HP_EPOCHS=10\u001b[0m\n",
      "\u001b[34mSM_RESOURCE_CONFIG={\"current_host\":\"algo-1\",
\"hosts\":[\"algo-1\"],\"network_interface_name\":\"eth0\"}
```

```
\u001b[0m\n",
      "\u001b[34mSM_HP_HIDDEN_DIM=200\u001b[0m\n",
      "\u001b[34mSM_OUTPUT_DATA_DIR=/opt/ml/output/data\u001b[0m\n",
      "\u001b[34mSM_INPUT_DATA_CONFIG={\"training\":
{\"RecordWrapperType\":\"None\",\"S3DistributionType\":
\"FullyReplicated\",\"TrainingInputMode\":\"File\"}}\u001b[0m\n",
      "\u001b[34mSM_HOSTS=[\"algo-1\"]\u001b[0m\n",
      "\u001b[34mSM_USER_ARGS=[\"--epochs\",\"10\",\"--hidden_dim\",
\"200\"]\u001b[0m\n",
      "\u001b[34mSM_NUM_GPUS=1\u001b[0m\n",
      "\u001b[34mSM_CURRENT_HOST=algo-1\u001b[0m\n",
      "\u001b[34mSM_USER_ENTRY_POINT=train.py\u001b[0m\n",
      "\u001b[34mSM_LOG_LEVEL=20\u001b[0m\n",
      "\u001b[34mSM_HPS={\"epochs\":10,\"hidden_dim\":200}\n",
      "\u001b[0m\n",
      "\u001b[34mInvoking script with the following command:\n",
      "\u001b[0m\n",
      "\u001b[34m/usr/bin/python -m train --epochs 10 --hidden_dim
200\n",
      "\n",
      "\u001b[0m\n",
      "\u001b[34mUsing device cuda.\u001b[0m\n",
      "\u001b[34mGet train data loader.\u001b[0m\n"
     ]
    },
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "\u001b[34mModel loaded with embedding_dim 32, hidden_dim 200,
vocab_size 5000.\u001b[0m\n",
      "\u001b[34mEpoch: 1, BCELoss: 0.6750282353284408\u001b[0m\n",
      "\u001b[34mEpoch: 2, BCELoss: 0.6029751568424458\u001b[0m\n",
      "\u001b[34mEpoch: 3, BCELoss: 0.5077848841949385\u001b[0m\n",
      "\u001b[34mEpoch: 4, BCELoss: 0.4360883746828352\u001b[0m\n",
      "\u001b[34mEpoch: 5, BCELoss: 0.3896679634950599\u001b[0m\n",
      "\u001b[34mEpoch: 6, BCELoss: 0.3607030930567606\u001b[0m\n",
      "\u001b[34mEpoch: 7, BCELoss: 0.3470715989871901\u001b[0m\n",
      "\u001b[34mEpoch: 8, BCELoss: 0.3248112451057044\u001b[0m\n",
      "\u001b[34mEpoch: 9, BCELoss: 0.3167588814180724\u001b[0m\n",
      "\n",
      "2020-05-06 13:45:13 Uploading - Uploading generated training
model\n",
      "2020-05-06 13:45:13 Completed - Training job completed\n",
      "\u001b[34mEpoch: 10, BCELoss: 0.3138456370558056\u001b[0m\n",
      "\u001b[34m2020-05-06 13:45:02,312 sagemaker-containers INFO
Reporting training SUCCESS\u001b[0m\n",
      "Training seconds: 283\n",
      "Billable seconds: 283\n"
     ]
```

```
    }
   ],
   "source": [
    "estimator.fit({'training': input_data})"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 5: Testing the model\n",
    "\n",
    "As mentioned at the top of this notebook, we will be testing this model by first deploying it and then sending the testing data to the deployed endpoint. We will do this so that we can make sure that the deployed model is working correctly.\n",
    "\n",
    "## Step 6: Deploy the model for testing\n",
    "\n",
    "Now that we have trained our model, we would like to test it to see how it performs. Currently our model takes input of the form `review_length, review[500]` where `review[500]` is a sequence of `500` integers which describe the words present in the review, encoded using `word_dict`. Fortunately for us, SageMaker provides built-in inference code for models with simple inputs such as this.\n",
    "\n",
    "There is one thing that we need to provide, however, and that is a function which loads the saved model. This function must be called `model_fn()` and takes as its only parameter a path to the directory where the model artifacts are stored. This function must also be present in the python file which we specified as the entry point. In our case the model loading function has been provided and so no changes need to be made.\n",
    "\n",
    "**NOTE**: When the built-in inference code is run it must import the `model_fn()` method from the `train.py` file. This is why the training code is wrapped in a main guard ( ie, `if __name__ == '__main__':` )\n",
    "\n",
    "Since we don't need to change anything in the code that was uploaded during training, we can simply deploy the current model as-is.\n",
    "\n",
    "**NOTE:** When deploying a model you are asking SageMaker to launch an compute instance that will wait for data to be sent to it. As a result, this compute instance will continue to run until *you* shut it down. This is important to know since the cost of a deployed endpoint depends on how long it has been running for.\n",
    "\n",
    "In other words **If you are no longer using a deployed endpoint,
```

```
shut it down!**\n",
    "\n",
    "**TODO:** Deploy the trained model."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 28,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "----------!"
     ]
    }
   ],
   "source": [
    "# TODO: Deploy the trained model\n",
    "predictor = estimator.deploy(initial_instance_count = 1,
instance_type = 'ml.m4.xlarge')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 7 - Use the model for testing\n",
    "\n",
    "Once deployed, we can read in the test data and send it off to
our deployed model to get some results. Once we collect all of the
results we can determine how accurate our model is."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 29,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_X = pd.concat([pd.DataFrame(test_X_len),
pd.DataFrame(test_X)], axis=1)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 30,
   "metadata": {},
   "outputs": [],
```

```
   "source": [
    "# We split the data into chunks and send each chunk seperately,
accumulating the results.\n",
    "\n",
    "def predict(data, rows=512):\n",
    "    split_array = np.array_split(data, int(data.shape[0] /
float(rows) + 1))\n",
    "    predictions = np.array([])\n",
    "    for array in split_array:\n",
    "        predictions = np.append(predictions,
predictor.predict(array))\n",
    "    \n",
    "    return predictions"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 31,
   "metadata": {},
   "outputs": [],
   "source": [
    "predictions = predict(test_X.values)\n",
    "predictions = [round(num) for num in predictions]"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 32,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "0.846"
      ]
     },
     "execution_count": 32,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "from sklearn.metrics import accuracy_score\n",
    "accuracy_score(test_y, predictions)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
```

```
    "**Question:** How does this model compare to the XGBoost model
you created earlier? Why might these two models perform differently on
this dataset? Which do *you* think is better for sentiment analysis?"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "**Answer:**  This pytorch model performs slightly better than the
xgboost model. Could be because of the large data set."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### (TODO) More testing\n",
    "\n",
    "We now have a trained model which has been deployed and which we
can send processed reviews to and which returns the predicted
sentiment. However, ultimately we would like to be able to send our
model an unprocessed review. That is, we would like to send the review
itself as a string. For example, suppose we wish to send the following
review to our model."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 34,
   "metadata": {},
   "outputs": [],
   "source": [
    "test_review = 'The simplest pleasures in life are the best, and
this film is one of them. Combining a rather basic storyline of love
and adventure this movie transcends the usual weekend fair with wit
and unmitigated charm.'"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
```

```
    "The question we now need to answer is, how do we send this review to our model?\n",
    "\n",
    "Recall in the first section of this notebook we did a bunch of data processing to the IMDb dataset. In particular, we did two specific things to the provided reviews.\n",
    " - Removed any html tags and stemmed the input\n",
    " - Encoded the review as a sequence of integers using `word_dict`\n",
    " \n",
    "In order process the review we will need to repeat these two steps.\n",
    "\n",
    "**TODO**: Using the `review_to_words` and `convert_and_pad` methods from section one, convert `test_review` into a numpy array `test_data` suitable to send to our model. Remember that our model expects input of the form `review_length, review[500]`."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 35,
   "metadata": {},
   "outputs": [],
   "source": [
    "# TODO: Convert test_review into a form usable by the model and save the results in test_data\n",
    "test_data = [np.array(convert_and_pad(word_dict, review_to_words(test_review))[0])]"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Now that we have processed the review, we can send the resulting array to our model to predict the sentiment of the review."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 36,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "array(0.51195765, dtype=float32)"
      ]
     },
```

```
     "execution_count": 36,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "predictor.predict(test_data)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Since the return value of our model is close to `1`, we can be certain that the review we submitted is positive."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Delete the endpoint\n",
    "\n",
    "Of course, just like in the XGBoost notebook, once we've deployed an endpoint it continues to run until we tell it to shut down. Since we are done using our endpoint for now, we can delete it."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 37,
   "metadata": {},
   "outputs": [],
   "source": [
    "estimator.delete_endpoint()"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 6 (again) - Deploy the model for the web app\n",
    "\n",
    "Now that we know that our model is working, it's time to create some custom inference code so that we can send the model a review which has not been processed and have it determine the sentiment of the review.\n",
    "\n",
    "As we saw above, by default the estimator which we created, when deployed, will use the entry script and directory which we provided
```

```
when creating the model. However, since we now wish to accept a string as input and our model expects a processed review, we need to write some custom inference code.\n",
    "\n",
    "We will store the code that we write in the `serve` directory. Provided in this directory is the `model.py` file that we used to construct our model, a `utils.py` file which contains the `review_to_words` and `convert_and_pad` pre-processing functions which we used during the initial data processing, and `predict.py`, the file which will contain our custom inference code. Note also that `requirements.txt` is present which will tell SageMaker what Python libraries are required by our custom inference code.\n",
    "\n",
    "When deploying a PyTorch model in SageMaker, you are expected to provide four functions which the SageMaker inference container will use.\n",
    " - `model_fn`: This function is the same function that we used in the training script and it tells SageMaker how to load our model.\n",
    " - `input_fn`: This function receives the raw serialized input that has been sent to the model's endpoint and its job is to de-serialize and make the input available for the inference code.\n",
    " - `output_fn`: This function takes the output of the inference code and its job is to serialize this output and return it to the caller of the model's endpoint.\n",
    " - `predict_fn`: The heart of the inference script, this is where the actual prediction is done and is the function which you will need to complete.\n",
    "\n",
    "For the simple website that we are constructing during this project, the `input_fn` and `output_fn` methods are relatively straightforward. We only require being able to accept a string as input and we expect to return a single value as output. You might imagine though that in a more complex application the input or output may be image data or some other binary data which would require some effort to serialize.\n",
    "\n",
    "### (TODO) Writing inference code\n",
    "\n",
    "Before writing our custom inference code, we will begin by taking a look at the code which has been provided."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 38,
   "metadata": {},
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
```

```
     "text": [
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36margparse\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mjson\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mos\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mpickle\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36msys\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36msagemaker_containers\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mpandas\u001b[39;49;00m
\u001b[34mas\u001b[39;49;00m
\u001b[04m\u001b[36mpd\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mnumpy\u001b[39;49;00m \u001b[34mas\u001b[39;49;00m
\u001b[04m\u001b[36mnp\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mtorch\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mtorch.nn\u001b[39;49;00m
\u001b[34mas\u001b[39;49;00m
\u001b[04m\u001b[36mnn\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mtorch.optim\u001b[39;49;00m
\u001b[34mas\u001b[39;49;00m
\u001b[04m\u001b[36moptim\u001b[39;49;00m\r\n",
      "\u001b[34mimport\u001b[39;49;00m
\u001b[04m\u001b[36mtorch.utils.data\u001b[39;49;00m\r\n",
      "\r\n",
      "\u001b[34mfrom\u001b[39;49;00m
\u001b[04m\u001b[36mmodel\u001b[39;49;00m
\u001b[34mimport\u001b[39;49;00m LSTMClassifier\r\n",
      "\r\n",
      "\u001b[34mfrom\u001b[39;49;00m
\u001b[04m\u001b[36mutils\u001b[39;49;00m
\u001b[34mimport\u001b[39;49;00m review_to_words,
convert_and_pad\r\n",
      "\r\n",
      "\u001b[34mdef\u001b[39;49;00m
\u001b[32mmodel_fn\u001b[39;49;00m(model_dir):\r\n",
      "    \u001b[33m\"\"\"Load the PyTorch model from the `model_dir`
directory.\"\"\"\u001b[39;49;00m\r\n",
      "
\u001b[34mprint\u001b[39;49;00m(\u001b[33m\"\u001b[39;49;00m\u001b[33m
Loading model.\u001b[39;49;00m\u001b[33m\"\u001b[39;49;00m)\r\n",
      "\r\n",
```

```
      "    \u001b[37m# First, load the parameters used to create the
model.\u001b[39;49;00m\r\n",
      "    model_info = {}\r\n",
      "    model_info_path = os.path.join(model_dir,
\u001b[33m'\u001b[39;49;00m\u001b[33mmodel_info.pth\u001b[39;49;00m\u0
01b[33m'\u001b[39;49;00m)\r\n",
      "    \u001b[34mwith\u001b[39;49;00m
\u001b[36mopen\u001b[39;49;00m(model_info_path,
\u001b[33m'\u001b[39;49;00m\u001b[33mrb\u001b[39;49;00m\u001b[33m'\u00
1b[39;49;00m) \u001b[34mas\u001b[39;49;00m f:\r\n",
      "        model_info = torch.load(f)\r\n",
      "\r\n",
      "
\u001b[34mprint\u001b[39;49;00m(\u001b[33m\"\u001b[39;49;00m\u001b[33m
model_info: {}
\u001b[39;49;00m\u001b[33m\"\u001b[39;49;00m.format(model_info))\r\n",
      "\r\n",
      "    \u001b[37m# Determine the device and construct the model.
\u001b[39;49;00m\r\n",
      "    device =
torch.device(\u001b[33m\"\u001b[39;49;00m\u001b[33mcuda\u001b[39;49;00
m\u001b[33m\"\u001b[39;49;00m \u001b[34mif\u001b[39;49;00m
torch.cuda.is_available() \u001b[34melse\u001b[39;49;00m
\u001b[33m\"\u001b[39;49;00m\u001b[33mcpu\u001b[39;49;00m\u001b[33m\"\
u001b[39;49;00m)\r\n",
      "    model =
LSTMClassifier(model_info[\u001b[33m'\u001b[39;49;00m\u001b[33membeddi
ng_dim\u001b[39;49;00m\u001b[33m'\u001b[39;49;00m],
model_info[\u001b[33m'\u001b[39;49;00m\u001b[33mhidden_dim\u001b[39;49
;00m\u001b[33m'\u001b[39;49;00m],
model_info[\u001b[33m'\u001b[39;49;00m\u001b[33mvocab_size\u001b[39;49
;00m\u001b[33m'\u001b[39;49;00m])\r\n",
      "\r\n",
      "    \u001b[37m# Load the store model parameters.
\u001b[39;49;00m\r\n",
      "    model_path = os.path.join(model_dir,
\u001b[33m'\u001b[39;49;00m\u001b[33mmodel.pth\u001b[39;49;00m\u001b[3
3m'\u001b[39;49;00m)\r\n",
      "    \u001b[34mwith\u001b[39;49;00m
\u001b[36mopen\u001b[39;49;00m(model_path,
\u001b[33m'\u001b[39;49;00m\u001b[33mrb\u001b[39;49;00m\u001b[33m'\u00
1b[39;49;00m) \u001b[34mas\u001b[39;49;00m f:\r\n",
      "        model.load_state_dict(torch.load(f))\r\n",
      "\r\n",
      "    \u001b[37m# Load the saved word_dict.\u001b[39;49;00m\r\n",
      "    word_dict_path = os.path.join(model_dir,
\u001b[33m'\u001b[39;49;00m\u001b[33mword_dict.pkl\u001b[39;49;00m\u00
1b[33m'\u001b[39;49;00m)\r\n",
      "    \u001b[34mwith\u001b[39;49;00m
\u001b[36mopen\u001b[39;49;00m(word_dict_path,
```

```
\u001b[33m'\u001b[39;49;00m\u001b[33mrb\u001b[39;49;00m\u001b[33m'\u00
1b[39;49;00m) \u001b[34mas\u001b[39;49;00m f:\r\n",
      "        model.word_dict = pickle.load(f)\r\n",
      "\r\n",
      "    model.to(device).eval()\r\n",
      "\r\n",
      "
\u001b[34mprint\u001b[39;49;00m(\u001b[33m\"\u001b[39;49;00m\u001b[33m
Done loading model.\u001b[39;49;00m\u001b[33m\"\u001b[39;49;00m)\r\n",
      "    \u001b[34mreturn\u001b[39;49;00m model\r\n",
      "\r\n",
      "\u001b[34mdef\u001b[39;49;00m
\u001b[32minput_fn\u001b[39;49;00m(serialized_input_data,
content_type):\r\n",
      "
\u001b[34mprint\u001b[39;49;00m(\u001b[33m'\u001b[39;49;00m\u001b[33mD
eserializing the input data.
\u001b[39;49;00m\u001b[33m'\u001b[39;49;00m)\r\n",
      "    \u001b[34mif\u001b[39;49;00m content_type ==
\u001b[33m'\u001b[39;49;00m\u001b[33mtext/
plain\u001b[39;49;00m\u001b[33m'\u001b[39;49;00m:\r\n",
      "        data =
serialized_input_data.decode(\u001b[33m'\u001b[39;49;00m\u001b[33mutf-
8\u001b[39;49;00m\u001b[33m'\u001b[39;49;00m)\r\n",
      "        \u001b[34mreturn\u001b[39;49;00m data\r\n",
      "    \u001b[34mraise\u001b[39;49;00m
\u001b[36mException\u001b[39;49;00m(\u001b[33m'\u001b[39;49;00m\u001b[
33mRequested unsupported ContentType in content_type:
\u001b[39;49;00m\u001b[33m'\u001b[39;49;00m + content_type)\r\n",
      "\r\n",
      "\u001b[34mdef\u001b[39;49;00m
\u001b[32moutput_fn\u001b[39;49;00m(prediction_output, accept):\r\n",
      "
\u001b[34mprint\u001b[39;49;00m(\u001b[33m'\u001b[39;49;00m\u001b[33mS
erializing the generated output.
\u001b[39;49;00m\u001b[33m'\u001b[39;49;00m)\r\n",
      "    \u001b[34mreturn\u001b[39;49;00m
\u001b[36mstr\u001b[39;49;00m(prediction_output)\r\n",
      "\r\n",
      "\u001b[34mdef\u001b[39;49;00m
\u001b[32mpredict_fn\u001b[39;49;00m(input_data, model):\r\n",
      "
\u001b[34mprint\u001b[39;49;00m(\u001b[33m'\u001b[39;49;00m\u001b[33mI
nferring sentiment of input data.
\u001b[39;49;00m\u001b[33m'\u001b[39;49;00m)\r\n",
      "\r\n",
      "    device =
torch.device(\u001b[33m\"\u001b[39;49;00m\u001b[33mcuda\u001b[39;49;00
m\u001b[33m\"\u001b[39;49;00m \u001b[34mif\u001b[39;49;00m
torch.cuda.is_available() \u001b[34melse\u001b[39;49;00m
```

```
\u001b[33m\"\u001b[39;49;00m\u001b[33mcpu\u001b[39;49;00m\u001b[33m\"\
u001b[39;49;00m)\r\n",
      "    \r\n",
      "    \u001b[34mif\u001b[39;49;00m model.word_dict
\u001b[35mis\u001b[39;49;00m \u001b[36mNone\u001b[39;49;00m:\r\n",
      "        \u001b[34mraise\u001b[39;49;00m
\u001b[36mException\u001b[39;49;00m(\u001b[33m'\u001b[39;49;00m\u001b[
33mModel has not been loaded properly, no word_dict.
\u001b[39;49;00m\u001b[33m'\u001b[39;49;00m)\r\n",
      "    \r\n",
      "    \u001b[37m# TODO: Process input_data so that it is ready to
be sent to our model.\u001b[39;49;00m\r\n",
      "    \u001b[37m#       You should produce two variables:
\u001b[39;49;00m\r\n",
      "    \u001b[37m#         data_X   - A sequence of length 500
which represents the converted review\u001b[39;49;00m\r\n",
      "    \u001b[37m#         data_len - The length of the
review\u001b[39;49;00m\r\n",
      "    \r\n",
      "    words = review_to_words(input_data)\r\n",
      "    \r\n",
      "\r\n",
      "    data_X = convert_and_pad(model.word_dict, words)
[\u001b[34m0\u001b[39;49;00m]\r\n",
      "    data_len = convert_and_pad(model.word_dict, words)
[\u001b[34m1\u001b[39;49;00m]\r\n",
      "\r\n",
      "    \u001b[37m# Using data_X and data_len we construct an
appropriate input tensor. Remember\u001b[39;49;00m\r\n",
      "    \u001b[37m# that our model expects input data of the form
'len, review[500]'.\u001b[39;49;00m\r\n",
      "    data_pack = np.hstack((data_len, data_X))\r\n",
      "    data_pack = data_pack.reshape(\u001b[34m1\u001b[39;49;00m,
-\u001b[34m1\u001b[39;49;00m)\r\n",
      "    \r\n",
      "    data = torch.from_numpy(data_pack)\r\n",
      "    data = data.to(device)\r\n",
      "\r\n",
      "    \u001b[37m# Make sure to put the model into evaluation
mode\u001b[39;49;00m\r\n",
      "    model.eval()\r\n",
      "\r\n",
      "    \u001b[37m# TODO: Compute the result of applying the model
to the input data. The variable `result` should\u001b[39;49;00m\r\n",
      "    \u001b[37m#       be a numpy array which contains a single
integer which is either 1 or 0\u001b[39;49;00m\r\n",
      "    \r\n",
      "    \u001b[34mwith\u001b[39;49;00m torch.no_grad():\r\n",
      "        output = model.forward(data)\r\n",
      "\r\n",
```

```
      "    result = np.round(output.numpy())\r\n",
      "\r\n",
      "    \u001b[34mreturn\u001b[39;49;00m result\r\n"
     ]
    }
   ],
   "source": [
    "!pygmentize serve/predict.py"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "As mentioned earlier, the `model_fn` method is the same as the one provided in the training code and the `input_fn` and `output_fn` methods are very simple and your task will be to complete the `predict_fn` method. Make sure that you save the completed file as `predict.py` in the `serve` directory.\n",
    "\n",
    "**TODO**: Complete the `predict_fn()` method in the `serve/predict.py` file."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Deploying the model\n",
    "\n",
    "Now that the custom inference code has been written, we will create and deploy our model. To begin with, we need to construct a new PyTorchModel object which points to the model artifacts created during training and also points to the inference code that we wish to use. Then we can call the deploy method to launch the deployment container.\n",
    "\n",
    "**NOTE**: The default behaviour for a deployed PyTorch model is to assume that any input passed to the predictor is a `numpy` array. In our case we want to send a string so we need to construct a simple wrapper around the `RealTimePredictor` class to accomodate simple strings. In a more complicated situation you may want to provide a serialization object, for example if you wanted to sent image data."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 39,
   "metadata": {},
   "outputs": [
```

```
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "-------------!"
     ]
    }
   ],
   "source": [
    "from sagemaker.predictor import RealTimePredictor\n",
    "from sagemaker.pytorch import PyTorchModel\n",
    "\n",
    "class StringPredictor(RealTimePredictor):\n",
    "    def __init__(self, endpoint_name, sagemaker_session):\n",
    "        super(StringPredictor, self).__init__(endpoint_name, 
sagemaker_session, content_type='text/plain')\n",
    "\n",
    "model = PyTorchModel(model_data=estimator.model_data,\n",
    "                     role = role,\n",
    "                     framework_version='0.4.0',\n",
    "                     entry_point='predict.py',\n",
    "                     source_dir='serve',\n",
    "                     predictor_cls=StringPredictor)\n",
    "predictor = model.deploy(initial_instance_count=1, 
instance_type='ml.m4.xlarge')"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Testing the model\n",
    "\n",
    "Now that we have deployed our model with the custom inference 
code, we should test to see if everything is working. Here we test our 
model by loading the first `250` positive and negative reviews and 
send them to the endpoint, then collect the results. The reason for 
only sending some of the data is that the amount of time it takes for 
our model to process the input and then perform inference is quite 
long and so testing the entire data set would be prohibitive."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 42,
   "metadata": {},
   "outputs": [],
   "source": [
    "import glob\n",
    "\n",
```

```
    "def test_reviews(data_dir='../data/aclImdb', stop=250):\n",
    "    \n",
    "    results = []\n",
    "    ground = []\n",
    "    \n",
    "    # We make sure to test both positive and negative reviews
\n",
    "    for sentiment in ['pos', 'neg']:\n",
    "        \n",
    "        path = os.path.join(data_dir, 'test', sentiment, '*.txt')
\n",
    "        files = glob.glob(path)\n",
    "        \n",
    "        files_read = 0\n",
    "        \n",
    "        print('Starting ', sentiment, ' files')\n",
    "        \n",
    "        # Iterate through the files and send them to the
predictor\n",
    "        for f in files:\n",
    "            with open(f) as review:\n",
    "                # First, we store the ground truth (was the
review positive or negative)\n",
    "                if sentiment == 'pos':\n",
    "                    ground.append(1)\n",
    "                else:\n",
    "                    ground.append(0)\n",
    "                # Read in the review and convert to 'utf-8' for
transmission via HTTP\n",
    "                review_input = review.read().encode('utf-8')\n",
    "                # Send the review to the predictor and store the
results\n",
    "
results.append(int(float(predictor.predict(review_input))))\n",
    "                \n",
    "            # Sending reviews to our endpoint one at a time takes
a while so we\n",
    "            # only send a small number of reviews\n",
    "            files_read += 1\n",
    "            if files_read == stop:\n",
    "                break\n",
    "            \n",
    "    return ground, results"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 43,
   "metadata": {},
   "outputs": [
```

```
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Starting  pos  files\n",
      "Starting  neg  files\n"
     ]
    }
   ],
   "source": [
    "ground, results = test_reviews()"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 44,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "0.836"
      ]
     },
     "execution_count": 44,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "from sklearn.metrics import accuracy_score\n",
    "accuracy_score(ground, results)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "As an additional test, we can try sending the `test_review` that
we looked at earlier."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 45,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
```

```
       "b'1.0'"
      ]
     },
     "execution_count": 45,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "predictor.predict(test_review)"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Now that we know our endpoint is working as expected, we can set up the web page that will interact with it. If you don't have time to finish the project now, make sure to skip down to the end of this notebook and shut down your endpoint. You can deploy it again when you come back."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 7 (again): Use the model for the web app\n",
    "\n",
    "> **TODO:** This entire section and the next contain tasks for you to complete, mostly using the AWS console.\n",
    "\n",
    "So far we have been accessing our model endpoint by constructing a predictor object which uses the endpoint and then just using the predictor object to perform inference. What if we wanted to create a web app which accessed our model? The way things are set up currently makes that not possible since in order to access a SageMaker endpoint the app would first have to authenticate with AWS using an IAM role which included access to SageMaker endpoints. However, there is an easier way! We just need to use some additional AWS services.\n",
    "\n",
    "<img src=\"Web App Diagram.svg\">\n",
    "\n",
    "The diagram above gives an overview of how the various services will work together. On the far right is the model which we trained above and which is deployed using SageMaker. On the far left is our web app that collects a user's movie review, sends it off and expects a positive or negative sentiment in return.\n",
    "\n",
    "In the middle is where some of the magic happens. We will
```

```
construct a Lambda function, which you can think of as a straightforward Python function that can be executed whenever a specified event occurs. We will give this function permission to send and recieve data from a SageMaker endpoint.\n",
    "\n",
    "Lastly, the method we will use to execute the Lambda function is a new endpoint that we will create using API Gateway. This endpoint will be a url that listens for data to be sent to it. Once it gets some data it will pass that data on to the Lambda function and then return whatever the Lambda function returns. Essentially it will act as an interface that lets our web app communicate with the Lambda function.\n",
    "\n",
    "### Setting up a Lambda function\n",
    "\n",
    "The first thing we are going to do is set up a Lambda function. This Lambda function will be executed whenever our public API has data sent to it. When it is executed it will receive the data, perform any sort of processing that is required, send the data (the review) to the SageMaker endpoint we've created and then return the result.\n",
    "\n",
    "#### Part A: Create an IAM Role for the Lambda function\n",
    "\n",
    "Since we want the Lambda function to call a SageMaker endpoint, we need to make sure that it has permission to do so. To do this, we will construct a role that we can later give the Lambda function.\n",
    "\n",
    "Using the AWS Console, navigate to the **IAM** page and click on **Roles**. Then, click on **Create role**. Make sure that the **AWS service** is the type of trusted entity selected and choose **Lambda** as the service that will use this role, then click **Next: Permissions**.\n",
    "\n",
    "In the search box type `sagemaker` and select the check box next to the **AmazonSageMakerFullAccess** policy. Then, click on **Next: Review**.\n",
    "\n",
    "Lastly, give this role a name. Make sure you use a name that you will remember later on, for example `LambdaSageMakerRole`. Then, click on **Create role**.\n",
    "\n",
    "#### Part B: Create a Lambda function\n",
    "\n",
    "Now it is time to actually create the Lambda function.\n",
    "\n",
    "Using the AWS Console, navigate to the AWS Lambda page and click on **Create a function**. When you get to the next page, make sure that **Author from scratch** is selected. Now, name your Lambda function, using a name that you will remember later on, for example `sentiment_analysis_func`. Make sure that the **Python 3.6** runtime
```

```
is selected and then choose the role that you created in the previous part. Then, click on **Create Function**.\n",
    "\n",
    "On the next page you will see some information about the Lambda function you've just created. If you scroll down you should see an editor in which you can write the code that will be executed when your Lambda function is triggered. In our example, we will use the code below. \n",
    "\n",
    "```python\n",
    "# We need to use the low-level library to interact with SageMaker since the SageMaker API\n",
    "# is not available natively through Lambda.\n",
    "import boto3\n",
    "\n",
    "def lambda_handler(event, context):\n",
    "\n",
    "    # The SageMaker runtime is what allows us to invoke the endpoint that we've created.\n",
    "    runtime = boto3.Session().client('sagemaker-runtime')\n",
    "\n",
    "    # Now we use the SageMaker runtime to invoke our endpoint, sending the review we were given\n",
    "    response = runtime.invoke_endpoint(EndpointName = '**ENDPOINT NAME HERE**',    # The name of the endpoint we created\n",
    "                                       ContentType = 'text/plain',                 # The data format that is expected\n",
    "                                       Body = event['body'])                       # The actual review\n",
    "\n",
    "    # The response is an HTTP response whose body contains the result of our inference\n",
    "    result = response['Body'].read().decode('utf-8')\n",
    "\n",
    "    return {\n",
    "        'statusCode' : 200,\n",
    "        'headers' : { 'Content-Type' : 'text/plain', 'Access-Control-Allow-Origin' : '*' },\n",
    "        'body' : result\n",
    "    }\n",
    "```\n",
    "\n",
    "Once you have copy and pasted the code above into the Lambda code editor, replace the `**ENDPOINT NAME HERE**` portion with the name of the endpoint that we deployed earlier. You can determine the name of the endpoint using the code cell below."
   ]
  },
  {
   "cell_type": "code",
```

```
   "execution_count": 46,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "text/plain": [
       "'sagemaker-pytorch-2020-05-06-14-05-59-180'"
      ]
     },
     "execution_count": 46,
     "metadata": {},
     "output_type": "execute_result"
    }
   ],
   "source": [
    "predictor.endpoint"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Once you have added the endpoint name to the Lambda function, click on **Save**. Your Lambda function is now up and running. Next we need to create a way for our web app to execute the Lambda function.\n",
    "\n",
    "### Setting up API Gateway\n",
    "\n",
    "Now that our Lambda function is set up, it is time to create a new API using API Gateway that will trigger the Lambda function we have just created.\n",
    "\n",
    "Using AWS Console, navigate to **Amazon API Gateway** and then click on **Get started**.\n",
    "\n",
    "On the next page, make sure that **New API** is selected and give the new api a name, for example, `sentiment_analysis_api`. Then, click on **Create API**.\n",
    "\n",
    "Now we have created an API, however it doesn't currently do anything. What we want it to do is to trigger the Lambda function that we created earlier.\n",
    "\n",
    "Select the **Actions** dropdown menu and click **Create Method**. A new blank method will be created, select its dropdown menu and select **POST**, then click on the check mark beside it.\n",
    "\n",
    "For the integration point, make sure that **Lambda Function** is selected and click on the **Use Lambda Proxy integration**. This
```

```
option makes sure that the data that is sent to the API is then sent directly to the Lambda function with no processing. It also means that the return value must be a proper response object as it will also not be processed by API Gateway.\n",
    "\n",
    "Type the name of the Lambda function you created earlier into the **Lambda Function** text entry box and then click on **Save**. Click on **OK** in the pop-up box that then appears, giving permission to API Gateway to invoke the Lambda function you created.\n",
    "\n",
    "The last step in creating the API Gateway is to select the **Actions** dropdown and click on **Deploy API**. You will need to create a new Deployment stage and name it anything you like, for example `prod`.\n",
    "\n",
    "You have now successfully set up a public API to access your SageMaker model. Make sure to copy or write down the URL provided to invoke your newly created public API as this will be needed in the next step. This URL can be found at the top of the page, highlighted in blue next to the text **Invoke URL**."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "## Step 4: Deploying our web app\n",
    "\n",
    "Now that we have a publicly available API, we can start using it in a web app. For our purposes, we have provided a simple static html file which can make use of the public api you created earlier.\n",
    "\n",
    "In the `website` folder there should be a file called `index.html`. Download the file to your computer and open that file up in a text editor of your choice. There should be a line which contains **\\*\\*REPLACE WITH PUBLIC API URL\\*\\***. Replace this string with the url that you wrote down in the last step and then save the file.\n",
    "\n",
    "Now, if you open `index.html` on your local computer, your browser will behave as a local web server and you can use the provided site to interact with your SageMaker model.\n",
    "\n",
    "If you'd like to go further, you can host this html file anywhere you'd like, for example using github or hosting a static site on Amazon's S3. Once you have done this you can share the link with anyone you'd like and have them play with it too!\n",
    "\n",
    "> **Important Note** In order for the web app to communicate with the SageMaker endpoint, the endpoint has to actually be deployed and
```

```
running. This means that you are paying for it. Make sure that the endpoint is running when you want to use the web app but that you shut it down when you don't need it, otherwise you will end up with a surprisingly large AWS bill.\n",
    "\n",
    "**TODO:** Make sure that you include the edited `index.html` file in your project submission."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Now that your web app is working, trying playing around with it and see how well it works.\n",
    "\n",
    "**Question**: Give an example of a review that you entered into your web app. What was the predicted sentiment of your example review?"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "**Answer:** My test review - The movie was boring at start. The protagonist did a good job and the story was quite inspiring.\n",
    "\n",
    "Predicted Sentiment : Your review was POSITIVE!\n"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Delete the endpoint\n",
    "\n",
    "Remember to always shut down your endpoint if you are no longer using it. You are charged for the length of time that the endpoint is running so if you forget and leave it on you could end up with an unexpectedly large bill."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 47,
   "metadata": {},
   "outputs": [],
   "source": [
    "predictor.delete_endpoint()"
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "conda_pytorch_p36",
   "language": "python",
   "name": "conda_pytorch_p36"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.6.5"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 2
}
```