Creating a Sentiment Analysis Web App

Using PyTorch and SageMaker

Deep Learning Nanodegree Program | Deployment

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.

Instructions

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!

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.

> **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.

General Outline

Recall the general outline for SageMaker projects using a notebook instance.

1. Download or otherwise retrieve the data.
2. Process / Prepare the data.
3. Upload the processed data to S3.
4. Train a chosen model.
5. Test the trained model (typically using a batch transform job).
6. Deploy the trained model.
7. Use the deployed model.

model is working correctly before moving forward.

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.

```
# Make sure that we use SageMaker 1.x
!pip install sagemaker==1.72.0
```

```
Requirement already satisfied: sagemaker==1.72.0 in /home/ec2-user/anaconda3/e
Requirement already satisfied: protobuf3-to-dict>=0.1.5 in /home/ec2-user/anac
Requirement already satisfied: scipy>=0.19.0 in /home/ec2-user/anaconda3/envs/
Requirement already satisfied: packaging>=20.0 in /home/ec2-user/anaconda3/env
Requirement already satisfied: importlib-metadata>=1.4.0 in /home/ec2-user/ana
Requirement already satisfied: smdebug-rulesconfig==0.1.4 in /home/ec2-user/an
Requirement already satisfied: numpy>=1.9.0 in /home/ec2-user/anaconda3/envs/p
Requirement already satisfied: boto3>=1.14.12 in /home/ec2-user/anaconda3/envs
Requirement already satisfied: protobuf>=3.1 in /home/ec2-user/anaconda3/envs/
Requirement already satisfied: botocore<1.22.0,>=1.21.28 in /home/ec2-user/ana
Requirement already satisfied: s3transfer<0.6.0,>=0.5.0 in /home/ec2-user/anac
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /home/ec2-user/anacon
Requirement already satisfied: urllib3<1.27,>=1.25.4 in /home/ec2-user/anacond
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /home/ec2-user/a
Requirement already satisfied: zipp>=0.5 in /home/ec2-user/anaconda3/envs/pyto
Requirement already satisfied: typing-extensions>=3.6.4 in /home/ec2-user/anac
Requirement already satisfied: pyparsing>=2.0.2 in /home/ec2-user/anaconda3/en
Requirement already satisfied: six>=1.9 in /home/ec2-user/anaconda3/envs/pytor
WARNING: You are using pip version 21.1.3; however, version 21.2.4 is availab
You should consider upgrading via the '/home/ec2-user/anaconda3/envs/pytorch_p
```

Step 1: Downloading the data

As in the XGBoost in SageMaker notebook, we will be using the IMDb dataset

> Maas, Andrew L., et al. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, 2011.

```
%mkdir ../data
```

```
2021-09-01 15:52:28 (47.4 MB/s) - '../data/aclImdb_v1.tar.gz' saved [84125825/
```

Step 2: Preparing and Processing the data

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.

```
import os
import glob

def read_imdb_data(data_dir='../data/aclImdb'):
    data = {}
    labels = {}

    for data_type in ['train', 'test']:
        data[data_type] = {}
        labels[data_type] = {}

        for sentiment in ['pos', 'neg']:
            data[data_type][sentiment] = []
            labels[data_type][sentiment] = []

            path = os.path.join(data_dir, data_type, sentiment, '*.txt')
            files = glob.glob(path)

            for f in files:
                with open(f) as review:
                    data[data_type][sentiment].append(review.read())
                    # Here we represent a positive review by '1' and a negative rev
                    labels[data_type][sentiment].append(1 if sentiment == 'pos' els

            assert len(data[data_type][sentiment]) == len(labels[data_type][sentime
                    "{}/{} data size does not match labels size".format(data_type,
```

```
def prepare_imdb_data(data, labels):
    """Prepare training and test sets from IMDb movie reviews."""

    #Combine positive and negative reviews and labels
    data_train = data['train']['pos'] + data['train']['neg']
    data_test = data['test']['pos'] + data['test']['neg']
    labels_train = labels['train']['pos'] + labels['train']['neg']
    labels_test = labels['test']['pos'] + labels['test']['neg']

    #Shuffle reviews and corresponding labels within training and test sets
    data_train, labels_train = shuffle(data_train, labels_train)
    data_test, labels_test = shuffle(data_test, labels_test)

    # Return a unified training data, test data, training labels, test labets
    return data_train, data_test, labels_train, labels_test


train_X, test_X, train_y, test_y = prepare_imdb_data(data, labels)
print("IMDb reviews (combined): train = {}, test = {}".format(len(train_X), len(tes
```

```
IMDb reviews (combined): train = 25000, test = 25000
```

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.

```
print(train_X[100])
print(train_y[100])
```

```
    return words
```

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.

```
# TODO: Apply review_to_words to a review (train_X[100] or any other review)
review_to_words(train_X[100])
```

```
['string',
 'attach',
 'one',
 'carlo',
 'mencia',
 'best',
 'perform',
 'date',
 'mencia',
 'known',
 'poke',
 'make',
 'fun',
```

```
 'workplac',
'terror',
'opinion',
'mel',
'gibson',
'passion',
'christ',
'argument',
'got',
'woman',
'regard',
```

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?

Answer: It remove morphological affixes from words, leaving only the word stem

```
        cache_data = dict(words_train=words_train, words_test=words_test,
                          labels_train=labels_train, labels_test=labels_test)
        with open(os.path.join(cache_dir, cache_file), "wb") as f:
            pickle.dump(cache_data, f)
        print("Wrote preprocessed data to cache file:", cache_file)
else:
    # Unpack data loaded from cache file
    words_train, words_test, labels_train, labels_test = (cache_data['words_tra
            cache_data['words_test'], cache_data['labels_train'], cache_data['l

return words_train, words_test, labels_train, labels_test
```

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.

[ ] ↳ *2 cells hidden*