Deploying a Sentiment Analysis Model

REVIEW | CODE REVIEW 2 | HISTORY

Meets Specifications

Hi, I am your new reviewer and this is the first time I am reviewing your submission.

You've done a great job taking into account all the suggestions from the previous reviewer and implementing all the requirements. 👏

Overall, I am really impressed with the amount of effort you've put into the project. You deserve applaud for your hardwork!

🎉 Finally, Congratulations on completing this project. You are one step closer to finishing your Nanodegree.

Wishing you good luck for all future projects 🎉

Some general suggestions

Use of assertions and Logging:

- Consider using Python assertions for sanity testing - assertions are great for catching bugs. This is especially true of a dynamically type-checked language like Python where a wrong variable type or shape can cause errors at runtime
- Logging is important for long-running applications. Logging done right produces a report that can be analyzed to debug errors and find crucial information. There could be different levels of logging or logging tags that can be used to filter messages most relevant to someone. Messages can be written to the terminal using print() or saved to file, for example using the Logger module. Sometimes it's worthwhile to catch and log exceptions during a long-running operation so that the operation itself is not aborted.

Debugging:

- Check out this guide on debugging in python

Reproducibility:

- Reproducibility is perhaps the biggest issue in machine learning right now. With so many moving parts present in the code (data, hyperparameters, etc) it is imperative that the instructions and code make it easy for anyone to get exactly the same results (just imagine debugging an ML pipeline where the data changes every time and so you cannot get the same result twice).
- Also consider using random seeds to make your data more reproducible.

Optimization and Profiling:

- Monitoring progress and debugging with Tensorboard: This tool can log detailed information about the model, data, hyperparameters, and more. Tensorboard can be used with Pytorch as well.
- Profiling with Pytorch: Pytorch's profiler can be used to break down profiling information by operations (convolution, pooling, batch norm) and identify performance bottlenecks. The performance traces can be viewed in the browser itself. The profiler is a great tool for quickly comparing GPU vs CPU speedups for example.

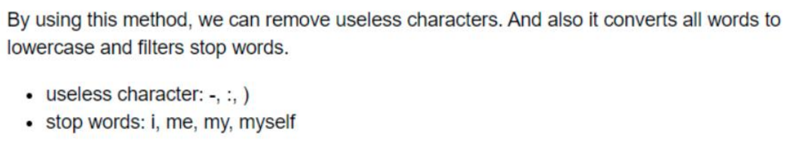
Files Submitted

✓ **The submission includes all required files, including notebook, python scripts, and html files.**

All files are included in the submission zip
✅ `Project Notebook`
✅ `index.html`
✅ `train.py`
✅ `predict.py`

Preparing and Processing Data

✓ **Answer describes what the pre-processing method does to a review.**



By using this method, we can remove useless characters. And also it converts all words to lowercase and filters stop words.

- useless character: -, :,)
- stop words: i, me, my, myself

You've correctly pointed out the modifications made by the pre-processing method to a review.

Note: The function gets rid of punctuations from the review using regular expressions.

```
text = re.sub(r"[^a-zA-Z0-9]", " ", text.lower())
```

In the above line of code, `re.sub` replaces all characters that are NOT alphabets or numbers with a space.
You can read more about re.sub here: https://docs.python.org/3/library/re.html#re.sub

✓ Answer describes how the processing methods are applied to the training and test data sets and what, if any, issues there may be.

> In order to preprocess our data, we must handle it in same way. If we use another ways to preprocess our data, result will be different.
>
> On the other hand, some contexts are smaller than 500, which mean, it will waste memory.

Good observation.
When pre-processing train and test data, we should use the same pre-processing **steps**. This is because the model that is trained is the same model on which we will test the data. Using same processing steps ensures both training and test data have similar representations.

However, it's important to note that we shouldn't accidentally use testing data while building word_dict in our case. That'll introduce data leakage and skew results.

✓ Notebook displays the five most frequently appearing words.

The 5 five most frequently appearing words are correctly displayed.

```
movi
film
one
like
time
```

✓ The `build_dict` method is implemented and constructs a valid word dictionary.

`build_dict` constructs a valid dictionary.

```
def build_dict(data, vocab_size = 5000):

    word_count = Counter(np.concatenate(data))

    sorted_words = sorted(word_count, key=word_count.get, reverse=True)

    word_dict = {word:idx + 2 for idx, word in enumerate(sorted_words[:vocab_size - 2])}


    return word_dict
```

Build and Train a PyTorch Model

✓ The train method is implemented and can be used to train the PyTorch model.

Good job at implementing the train method correctly.

For remembering the training steps I use the custom acronym: **ZOLS**
Z -> zero_grad()
O -> output (preds)
L -> loss
S -> optimizer.step()

You can create your own custom acronym to remember the training steps.

✓ The RNN is trained using SageMaker's supported PyTorch functionality.

`estimator.fit()` executed properly which is an indication that you implemented your `train()` method correctly.

```
/usr/bin/python -m train --epochs 10 --hidden_dim 200


Using device cuda.
Get train data loader.
Model loaded with embedding_dim 32, hidden_dim 200, vocab_size 5000.
Epoch: 1, BCELoss: 0.6694651927266803
Epoch: 2, BCELoss: 0.6279108755442561
Epoch: 3, BCELoss: 0.5255827611806442
Epoch: 4, BCELoss: 0.4446507357821173
Epoch: 5, BCELoss: 0.39971029515169104
Epoch: 6, BCELoss: 0.3722564730109001
Epoch: 7, BCELoss: 0.3325431730066027
Epoch: 8, BCELoss: 0.31196530376161846
Epoch: 9, BCELoss: 0.2923963726783285
Epoch: 10, BCELoss: 0.2803410291671753
2021-05-18 18:31:46,977 sagemaker-containers INFO     Reporting training SUCCES
S

2021-05-18 18:31:58 Uploading - Uploading generated training model
2021-05-18 18:31:58 Completed - Training job completed
```

Note - I verified the implemention in `train.py` too.

Deploy a Model for Testing

✓ The trained PyTorch model is successfully deployed.

The RNN model is successfully deployed to `ml.m4.xlarge` AWS instance.

Note: `m4` is a general purpose instance primarily used to host webapps that require significant computer while `p2` is a specialized instance with High Performance GPUs which are useful for ML tasks.

Use the Model for Testing

✓ Answer describes the differences between the RNN model and the XGBoost model and how they perform on the IMDB data.

> LSTM model has been developed for natural language processing. And now, we have been working on sentiment analysis which is a part of nlp. That is the reason why LSTM is succesful model than XGBoost

When choosing an algorithm we must pay close attention to our data. In our case the data consists of sentences where the context between words and the semantics of the overall sentence is very important. In such cases RNNs/LSTMs work better because they are able to generate a hidden state based on the sequence in which words appear. XGBoost cannot do that, therefore RNNs/LSTMs are **comparably** better at performing sentiment analysis

✓ The test review has been processed correctly and stored in the `test_data` variable.

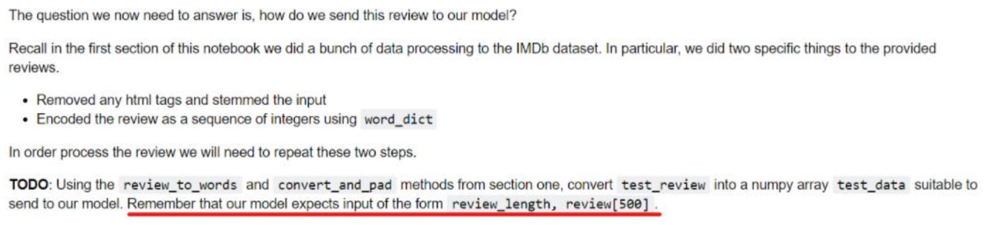
`predict` function executed properly and you've correctly processed the test review.

```
test_review_words = review_to_words(test_review)     # splits reviews to words
review_X, review_len = convert_and_pad(word_dict, test_review_words)   # pad review

data_pack = np.hstack((review_len, review_X))
data_pack = data_pack.reshape(1, -1)

test_data = torch.from_numpy(data_pack).to(device)
```

Good job passing the length of the review to predict function.



The question we now need to answer is, how do we send this review to our model?

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.

- Removed any html tags and stemmed the input
- Encoded the review as a sequence of integers using `word_dict`

In order process the review we will need to repeat these two steps.

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]`.

Suggestion:
Here's an alternate approach to this task -

```
review_list, review_length = convert_and_pad(word_dict, review_to_words(test_review))
test_data = np.array([np.array([review_length] + review_list)])
```

✓ The `predict_fn()` method in `serve/predict.py` has been implemented.

Nicely done! ✅

Deploying a Web App

✓ The model is deployed and the Lambda / API Gateway integration is complete so that the web app works (make sure to include your modified `index.html`).

```
https://6wvn2i8kui.execute-api.us-east-1.amazonaws.com/prod
```

AWS API is included in `index.html`

Underlying Mechanism is as follows:
On clicking the `Submit` button, the web app hits the AWS Lambda API and returns the model's prediction to the web app.

Overwhelmed with all the AWS lingo? Checkout this amazing website that explains all AWS Services in simple terms.

✓ Answer gives a sample review and the resulting predicted sentiment.

> My review was "the movie was horrible" and it said "your review was negative!" and also I entered second review as "the movie was great" it said "your review was positive!"

Awesome! The model correctly predicts the sentiment of the sample review.

⚠️ Suggestion: You could have also tried more complex review where the review may begin with one sentiment and then eventually make up for it and then present an opposite sentiment.
E.g. A review that starts negatively claiming that the movie was not as good as other movies by the director but overall it was excellent and well worth the time and money spent.

DOWNLOAD PROJECT

2 CODE REVIEW COMMENTS ›