‹ Return to Classroom

Deploying a Sentiment Analysis Model

REVIEW

CODE REVIEW 2

HISTORY

Requires Changes

1 specification requires changes

Dear Student,

You've done an excellent job at implementing the entire project! 👏

However, there are some changes that you need to make before you can pass the project.
As a reviewer I am required to follow Udacity's guidelines when evaluating projects and hence I had to mark some requirements that need more work from your end.

Please go through the review to see what changes you need to make.
Once you make the desired changes and resubmit the project, you will pass the project and get a step closer to finishing your nanodegree.

Wishing you good luck! 😄

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

- 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.

Make sure your submission contains:

- **The `SageMaker Project.ipynb` file with fully functional code, all code cells executed and displaying output, and all questions answered.**
- **An HTML or PDF export of the project notebook with the name `report.html` or `report.pdf`.**
- **The `train` folder with all provided files and the completed `train.py`.**
- **The `serve` folder with all provided files and the completed `predict.py`.**
- **The `website` folder with the edited `index.html` file.**

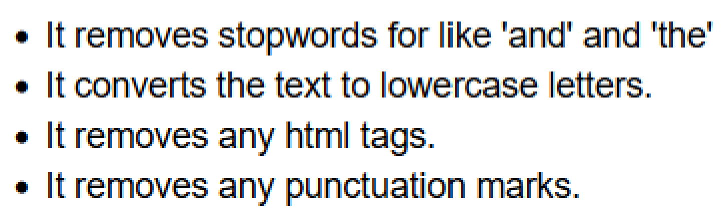
All files are included in the submission zip

✅ `Project Notebook`
✅ `index.html`
✅ `train.py`
✅ `predict.py`

Preparing and Processing Data

✓

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



- It removes stopwords for like 'and' and 'the'
- It converts the text to lowercase letters.
- It removes any html tags.
- It removes any punctuation marks.

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

✓

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

`build_dict` constructs a valid dictionary.

Suggestion: Here's a sample code snippet of an alternate implementation using `Counter` module.

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

    word_counter = collections.Counter()
    for sentence in data:
        word_counter.update(sentence)

    word_count = dict(word_counter.most_common())
    sorted_words = list(word_count.keys())

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

```
            word_dict[word] = idx + 2

    return word_dict
```

✓

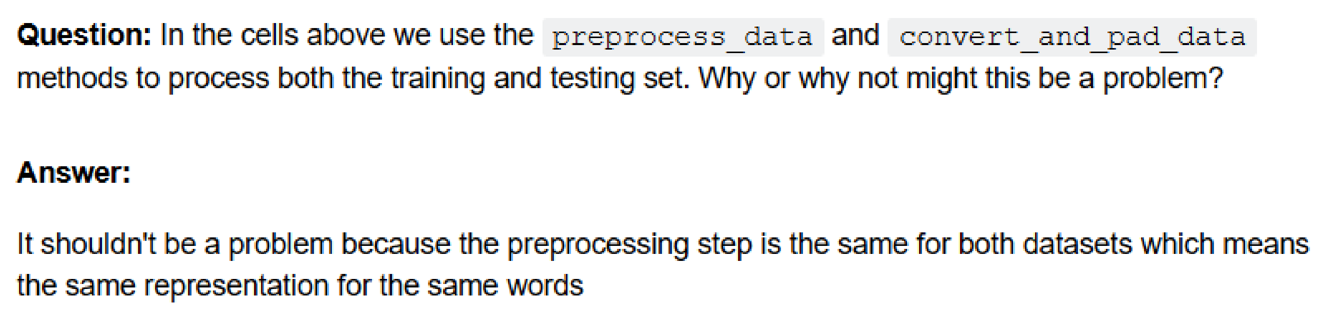
Notebook displays the five most frequently appearing words.

The 5 five most frequently appearing words are correctly displayed.

```
['movi', 'film', 'one', 'like', 'time']
```

✓

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



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?

Answer:

It shouldn't be a problem because the preprocessing step is the same for both datasets which means the same representation for the same words

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.

Build and Train the PyTorch Model

✓

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

```
def train(model, train_loader, epochs, optimizer, loss_fn, device):
    for epoch in range(1, epochs + 1):
        model.train()
        total_loss = 0
        for batch in train_loader:
            batch_X, batch_y = batch

            batch_X = batch_X.to(device)
            batch_y = batch_y.to(device)

            # TODO: Complete this train method to train the model provided.
            optimizer.zero_grad()
            preds=model(batch_X)
            loss=loss_fn(preds,batch_y)
            loss.backward()
            optimizer.step()

            total_loss += loss.data.item()
        print("Epoch: {}, BCELoss: {}".format(epoch, total_loss / len(train
_loader)))
```

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: 0.671411927865476
Epoch: 2, BCELoss: 0.5946382466627627
Epoch: 3, BCELoss: 0.5107948348230246
Epoch: 4, BCELoss: 0.43099653903318913
Epoch: 5, BCELoss: 0.37547219711907054
Epoch: 6, BCELoss: 0.34212268128687023
Epoch: 7, BCELoss: 0.3184543124267033
Epoch: 8, BCELoss: 0.3078041979852988
Epoch: 9, BCELoss: 0.3053491602138597

2021-12-17 15:37:47 Uploading - Uploading generated training modelEpoch:
10, BCELoss: 0.30380577426783895
2021-12-17 15:37:45,856 sagemaker-containers INFO     Reporting training
SUCCESS

2021-12-17 15:37:54 Completed - Training job completed
```

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

Deploy the 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.

Make sure your answer includes:

- The comparison between the two models

- Which model is better for sentiment analysis

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?

Answer: the perform quite the same, but i think with some parameter tuning the LSTM model would outperform the XGBoost model. because of the nature of both models as the XGBoost is a random tree calssifier while the LSTM model uses RNN that has memory and is designed to handle sequential data.

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.

While for shorter sequences such as IMDB data, this performance gap may not be significant, with larger datasets you'll see RNNs/LSTMs outperform other models.

The test review has been processed correctly and stored in the `test_data` variable. The test_data should contain two variables: review_len and review[500].

```
# TODO: Convert test_review into a form usable by the model and save the re
sults in test_data
test_data = np.array(convert_and_pad(word_dict,review_to_words(test_revie
w))[0]) [None,:]
```

Now that we have processed the review, we can send the resulting array to our model to predict the sentiment of the review.

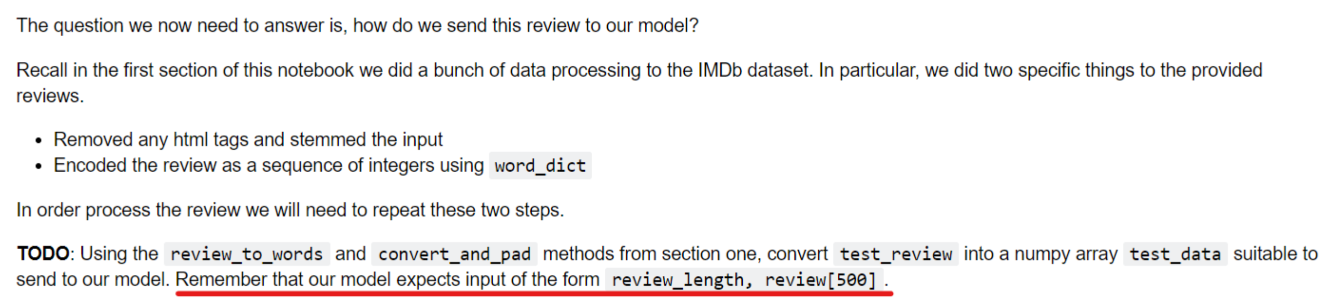
```
predictor.predict(test_data)
```

```
array(0.6553395, dtype=float32)
```

Oops! You forgot to pass the length of the review to the `predictor.predict()` function. Out of the two outputs that `convert_and_pad` function returns, you are only extracting the 1st output (0th index) which is the processed review. The 2nd output (1st index) is the review length. Notice the instructions in the project which I am pasting below for your convenience

---

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].**



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]`.

The ideal implementation would be

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

✓

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

- **The predict script should include both the data processing and the prediction.**
- **The processing should produce two variables: data_X and data_len.**

Nicely done! ✅

Deploying the 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://huhcqikotl.execute-api.us-east-1.amazonaws.com/default/sent_func
```

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.

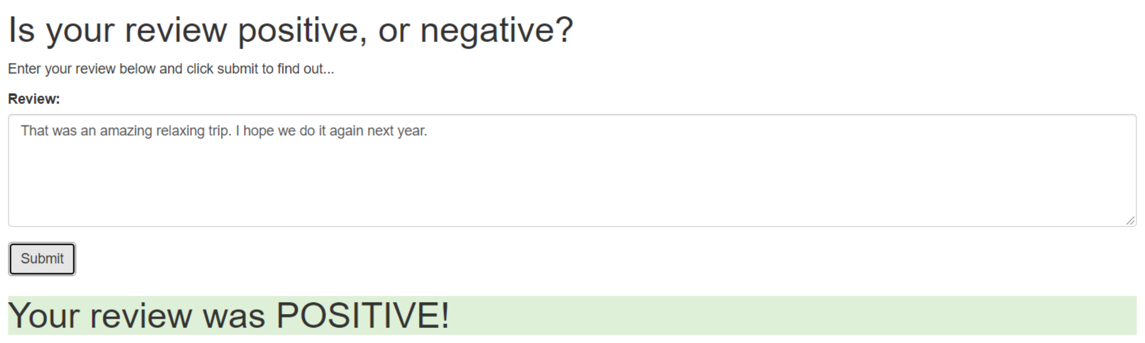
Overwhelmed with all the AWS lingo? Checkout this amazing website that explains

all AWS Services in simple terms.

✓

The answer includes a screenshot showing a sample review and the prediction.

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



Is your review positive, or negative?

Enter your review below and click submit to find out...

Review:

That was an amazing relaxing trip. I hope we do it again next year.

Your review was POSITIVE!

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.

RESUBMIT

DOWNLOAD PROJECT

2 CODE REVIEW COMMENTS ›

Best practices for your project resubmission

Ben shares 5 helpful tips to get you through revising and resubmitting your project.

Watch Video (3:01)

RETURN TO PATH