

[Return to Classroom](#)

Deploying a Sentiment Analysis Model

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Dear Student,

Well done 🌟 you have passed the sentiment analysis assignment with flying colours.

I appreciate the effort you have given from implementing preprocessing functionality to create an endpoint for testing and deployment. I like that you have trained the model locally as well as on the AWS server which shows that you can able to use AWS services very well. Good that you have used the P2 server for training the model and for testing you have used the m4 server.

congratulation on completing the deep learning nano degree. best of luck 🍀 for your future endeavours. and, happy learning 📖

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.

Good 🙌 all required files are available in the submission.

- ✓ notebook file
- ✓ train.py
- ✓ predict.py
- ✓ index.html

Preparing and Processing Data

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

Good 🙌 you have explained the other three preprocessing methods very well.

you can also explain more about the following topics with an example:

- Remove HTML tags
- Convert to lower case
- Split string into words
- Remove stopwords
- stemming

For reference, You can find the brief discussion for preprocessing of sentiment analysis [here](#)

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

Good 🙌 The implementation is a very basic programming approach.

To make the implementation more pythonic, you can implement the `build_dict` function as follows:

```
from collections import Counter

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
```

There are some other techniques that can help you to create a dictionary for specific use cases.

[link 1](#)

[link 2](#)

Notebook displays the five most frequently appearing words.

Great work 👍

you are able to get the top five frequent words as follows:

```
['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.

Good 🙌 you have explained that preprocess should be applied to both datasets.

but here, you have forgotten one more point to explain that is **data leakage**.

in the answer you can also add, the applied processes are correct because `word_dict` is generated by the training dataset only so there won't be a leaking issue.

for more information about data leakage [click here](#)

Build and Train the PyTorch Model

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

Great, the model is trained properly, as the loss is decreasing over the epochs.

Epoch: 1, BCELoss: 0.6946803569793701

Epoch: 2, BCELoss: 0.6842474460601806

Epoch: 3, BCELoss: 0.6756571650505065

Epoch: 4, BCELoss: 0.6659433007240295

Epoch: 5, BCELoss: 0.65362628698349

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

perfectly done 👍

I can see the logs which show that the model was trained properly.

```
2021-09-28 14:02:28,195 sagemaker-training-toolkit INFO Reporting training SUCCESS
```

Deploy the Model for Testing

The trained PyTorch model is successfully deployed.

it just one-liner code 😊 you remembered it very well

```
predictor = estimator.deploy(initial_instance_count = 1, instance_type = 'ml.m4.xlarge')
```

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

In this situation, yes XGboost performs similarly compared to LSTM but LSTM has an advanced structure in deep learning, so LSTM performs better overall.

There are some points which I want to highlight:

- input features for XGBoost are Bags of words and for the LSTM model, it is Word embedding. so the difference between the two features should be explained.
- The architectural difference between XGBoost and LSTM.
- Limitation of XGBoost should be mentioned.
- LSTM model complexity should be explained.

for more detail about LSTM and XGboost [click here](#)

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]`.

Good 🙌 you have used `convert_and_pad` and `review_to_words` correctly here.

```
test_review_words, test_review_len = convert_and_pad(word_dict, review_to_words(test_review))
```

Well done, you have considered the review length at the beginning of the test data.

```
test_data = np.array([[test_review_len] + test_review_words])
```

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`.

Good 🙌 you are using the same method for sagemaker

```
def predict_fn(input_data, model):
    print('Inferring sentiment of input data.')

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    if model.word_dict is None:
        raise Exception('Model has not been loaded properly, no word_dict.')

    # TODO: Process input_data so that it is ready to be sent to our model.
    #       You should produce two variables:
    #       data_X   - A sequence of length 500 which represents the converted review
    #       data_len - The length of the review

    data_X, data_len = convert_and_pad(model.word_dict, review_to_words(input_data))

    # Using data_X and data_len we construct an appropriate input tensor. Remember
    # that our model expects input data of the form 'len, review[500]'.
    data_pack = np.hstack((data_len, data_X))
    print('data_pack.shape= ', data_pack.shape) # shape =
```

```
data_pack = data_pack.reshape(1, -1)

data = torch.from_numpy(data_pack)
data = data.to(device)

# Make sure to put the model into evaluation mode
model.eval()

# TODO: Compute the result of applying the model to the input data. The variable
`result` should
#     be a numpy array which contains a single integer which is either 1 or 0

with torch.no_grad():
    output = model.forward(data)
    output = output.cpu()

result = np.round(output.numpy())

return result
```

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`).

Good! the link seems valid 👍

<https://vjrc5sqssk.execute-api.us-east-1.amazonaws.com/produces>

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

Perfect 🔥 You have attached test screenshots in the notebook.

 [DOWNLOAD PROJECT](#)

RETURN TO PATH

Rate this review

START