UDACITY

< Return to Classroom

Deploying a Sentiment Analysis Model

REVIEW

CODE REVIEW

HISTORY

Requires Changes

1 specification requires changes

Dear Student,

Great work 👏🏻 you can train and deploy the model on the AWS server.

I appreciate your efforts in creating a word dictionary for XGBoost training. Even, I also like that you have trained the LSTM model using the m4 server on AWS and the same server has been used to deploy the trained model.

But, during the testing on the local system, you have forgotten to add review length at the beginning of the test data.

So, please make changes as per the review comments and resubmit the assignment. till that time, 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 adding the last 500 words can help to summarize.

but here, you have forgotten one more point to explain that is **data leakage**.
in the answer, you can also add, that 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, you have implemented the train function correctly.

```
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.

            # Sets the gradients of all optimized torch.Tensor s to zero. se
e the following link
            # https://pytorch.org/docs/stable/generated/torch.optim.Optimize
r.zero_grad.html
            optimizer.zero_grad()

            # Predicting output of the model for the batch input batch_X and
calculating the loss
            predict = model.forward(batch_X)
            loss = loss_fn(predict, batch_y)

            # Backprogation
            # loss.backward() computes dloss/dx for every parameter x which
 has requires_grad=True.
            # see the following link for details
            # https://discuss.pytorch.org/t/what-does-the-backward-function-
do/9944
            loss.backward()
            #
            # Performs a single optimization step (parameter update). see th
e following link:
            # https://pytorch.org/docs/stable/generated/torch.optim.Optimize
r.step.html
            optimizer.step()

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

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

```
Epoch: 1, BCELoss: 0.694224739074707
Epoch: 2, BCELoss: 0.6844599485397339
Epoch: 3, BCELoss: 0.6764387369155884
Epoch: 4, BCELoss: 0.6679689288139343
Epoch: 5, BCELoss: 0.6579747676849366
```

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.

```
2022-06-18 20:37:59,649 sagemaker-containers INFO Reporting training SUCCESS
```

Deploy the Model for Testing

The trained PyTorch model is successfully deployed.

It is 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.
But if you read TODO which is as follows:

```
TODO: Using the review_to_words and convert_and_pad methods from section on
e, convert test_review into a NumPy array test_data suitable to send to our
 model. Remember that our model expects an input of the form review_length,
 review[500].
```

here you can read the last sentence which is explaining the expected input should contain form review_length, review[500]. but as per the code, I can see that you have only added review[500]. I have also given the solution as follows:

```
test_review_words = review_to_words(test_review)     # splits reviews to wor
ds
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)
test_data = test_data.to(device)
```

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 conve
rted review
    #         data_len - The length of the review


    review_to_words_output = review_to_words(input_data)
    data_X, data_len       = convert_and_pad(model.word_dict, review_to_word
s_output)
    # data_X = None
    #data_len = None

    # Using data_X and data_len we construct an appropriate input tensor. Re
member
    # that our model expects input data of the form 'len, review[500]'.
    data_pack = np.hstack((data_len, data_X))
    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():
         prediction = model.forward(data)

    result = np.round(prediction.numpy())
    #result = None

    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 👍

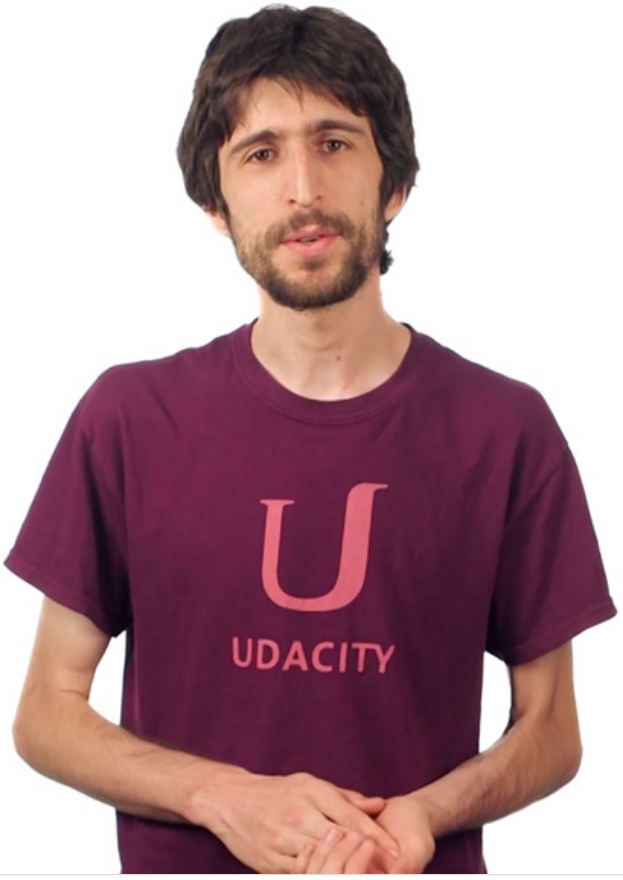
`https://kgn9wlasu0.execute-api.us-east-1.amazonaws.com/prod`

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

Perfect 🔥 you have attached a test screenshot in the notebook.

RESUBMIT

DOWNLOAD PROJECT



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

Rate this review

START