UDACITY

Logout

‹ Return to Classroom

Deploying a Sentiment Analysis Model

REVIEW | CODE REVIEW | HISTORY

Rate this review

START

Meets Specifications

Great job on project implementation!!!, you have correctly answered and implemented all "To Dos" in project notebook.
You have now gained good knowledge on deploying machine learning models in AWS cloud. This is very significant skill for ML professional as the success of overall project depends upon how ML application are productionalized after successfully building model.

Congratulations for finishing this project. All the best!!

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.

- SageMaker Project.html or SageMaker Project.ipynb
- index.html
- predict.py
- train.py

Preparing and Processing Data

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

Well done!! In addition to that it splits the string into words. Listed below are series of preprocessing steps done by this method.

1. Removes the html tags
2. Converts text to lower case.
3. Split string into words
4. Remove stopwords
5. Stems each word using porter stemmer.

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

Good work building word dictionary from sentences!!

Here's another approach for building word_count

```
word_count = {} # A dict storing the words that appear in the reviews along with how often they occur

    for review in data:
        for word in review:
            word_count[word] = word_count.get(word, 0) + 1


    sorted_words = sorted(word_count.keys(), key=lambda x: -word_count[x])
```

✓ Notebook displays the five most frequently appearing words.

Well done!! You have correctly evaluated five most frequent appearing words.

1: movi
2: film
3: one
4: like
5: time

```
# x = 0
for x,y in enumerate(list(word_dict.keys())):
    if x<5:
        print(y)
```

```
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 answer provided here also.
In addition, the idea is to prevent data leakage. Data leakage occurs when data from the training set leaks to the test set.

- preprocess_data is applied per record on both the training and test sets, so there is no issue coming from it.
- convert_and_pad_data doesn't cause an issue also because word_dict which is used to transform the reviews to integers was constructed using only the training data. If the test data was also used in creating word_dict, then predictions would be biased due to the data leakage. The test data is meant to be unseen data by the model.

Build and Train the PyTorch Model

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

Well done completing the train method to train the model provided.

Use torch.nn.utils.clip_grad_norm to keep the gradients within a specific range (clip). In RNNs the gradients tend to grow very large which may cause exploding gradient problem, clipping them helps to prevent this from happening .

```
        optimizer.zero_grad()
        output = model.forward(batch_X)

        loss = loss_fn(output,batch_y)

        loss.backward()
        optimizer.step()

        # TODO: Complete this train method to train the model provided.

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

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

Well done!! BCELoss decreases with subsequent epochs shows model has trained well.

```
Model loaded with embedding_dim 32, hidden_dim 80, vocab_size 5000.
Epoch: 1, BCELoss: 0.6808963819425933
Epoch: 2, BCELoss: 0.5955786948301354
Epoch: 3, BCELoss: 0.4767325855031305
Epoch: 4, BCELoss: 0.41110152371075687
```

Deploy the Model for Testing

✓ The trained PyTorch model is successfully deployed.

Good work deploying model to 'ml.m4.xlarge' instance.

```
# TODO: Deploy the trained model
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

Excellent!! RNN or LSTM are sequence based model as they store context of the sentence in the cell state. When further optimized and fine tuned they can outperform XGBoost for sentiment based classification.

✓ 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].

Well done preprocessing test_review data by applying review_to_words and convert_and_pad!!

```
# TODO: Convert test_review into a form usable by the model and save the results in test_data

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)
test_data = test_data.to(device)

#Source: udacity review
```

✓ 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.

Well done!! Same method has been used in script file to preprocess test review.

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

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`).

Well done creating lambda function and integrating with API endpoint.

```
<div class="container">
    <h1>Is your review positive, or negative?</h1>
    <p>Enter your review below and click submit to find out...</p>
    <form method="POST"
        action="https://y5rchw4tcj.execute-api.us-east-1.amazonaws.com/last"
        onsubmit="return submitForm(this);" >                    <!-- HERE IS WHERE YOU NEED TO ENTER THE API URL -->
        <div class="form-group">
            <label for="review">Review:</label>
            <textarea class="form-control"  rows="5" id="review">Please write your review here.</textarea>
        </div>
        <button type="submit" class="btn btn-default">Submit</button>
    </form>
    <h1 class="bg-success" id="result"></h1>
</div>
```

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

Good work on test prediction. I would recommend to run prediction on negative review as well. Consider review which are ambiguous and hard for model to predict sentiment specifically review which consists of sarcasm. This gives an idea about model shortcomings or limitations.

DOWNLOAD PROJECT

RETURN TO PATH