UDACITY

‹ Return to Classroom

Deploying a Sentiment Analysis Model

审阅 | 代码审阅 2 | HISTORY

需要修改

还需满足 2 个要求 变化

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 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.

Question: Above we mentioned that `review_to_words` method removes html formatting and allows us to tokenize the words found in a review, for example, converting *entertained* and *entertaining* into *entertain* so that they are treated as though they are the same word. What else, if anything, does this method do to the input?

Answer: After removing html formatting, the method replaces all uppercase letters with lowercase letters, splits the string into words and remove all insignificant words (stopwords).

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

Notebook displays the five most frequently appearing words.

Question: What are the five most frequently appearing (tokenized) words in the training set? Does it makes sense that these words appear frequently in the training set?

Answer: The five most frequently appearing words in our training data are "movi", "film", "one", "like" and "time". It looks reasonable since our training set is about movie reviews. It's normal for words related to movies and one's feelings to appear the most frequently.

```
# TODO: Use this space to determine the five most frequently appearing words in
the training set.
word_dict_list = list(word_dict.items())
print([word_dict_list[i] for i in range(5)])
```

```
[('movi', 2), ('film', 3), ('one', 4), ('like', 5), ('time', 6)]
```

The 5 five most frequently appearing words are correctly displayed.

```
[('movi', 2), ('film', 3), ('one', 4), ('like', 5), ('time', 6)]
```

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.

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

    return word_dict
```

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: The purpose of method `preprocess_data` is to cache the data so that we don't need to perform the processing step for a long time when we want to reuse this notebook. And the method `convert_and_pad_data` aims to convert the reviews into different integers with a fixed length in order to fit our future models.

But there may be a problem, in our model, we combined all of the infrequent words into a single category and labeled it as 1. In fact, many infrequent words may appear in a single review. Although we replace the infrequent words with 1, 1 still appears frequently and is considered the same word by the algorithm. I'm not sure if this affects the accuracy of the results. Increasing the number of word_dict may solve this problem, but the computation time will also increase.

⚠️ Good attempt but your answer doesn't address the actual question. The question is primarily concerned with the consequences of using the **same preprocessing methods** for training and testing data.

You've just described the functions themselves, but you are expected to discuss how using the same preprocessing methods for training and testing may or may not cause problems.

Build and Train a 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):
    """
    This is the training method that is called by the PyTorch training script. T
he parameters
    passed are as follows:
    model        - The PyTorch model that we wish to train.
    train_loader - The PyTorch DataLoader that should be used during training.
    epochs       - The total number of epochs to train for.
    optimizer    - The optimizer to use during training.
    loss_fn      - The loss function used for training.
    device       - Where the model and data should be loaded (gpu or cpu).
    """
    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()
            output = model.forward(batch_X)
            loss = loss_fn(output, batch_y)
            loss.backward()
            optimizer.step()

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

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.
load f
Model loaded with embedding_dim 32, hidden_dim 200, vocab_size 5000.
Epoch: 1, BCELoss: 0.6696704893696065
Epoch: 2, BCELoss: 0.5924909139166072
Epoch: 3, BCELoss: 0.5042590030602047
Epoch: 4, BCELoss: 0.4371729730343332
Epoch: 5, BCELoss: 0.3874118212534457
Epoch: 6, BCELoss: 0.35067193544640834
Epoch: 7, BCELoss: 0.3213530772802781
Epoch: 8, BCELoss: 0.299233760760755
Epoch: 9, BCELoss: 0.3211515454613433
Epoch: 10, BCELoss: 0.2744908490959479
2022-09-29 10:33:17,716 sagemaker-containers INFO     Reporting training SUCCES
S

2022-09-29 10:33:39 Uploading - Uploading generated training model
2022-09-29 10:33:39 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.p2.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: I got 0.86396 as the accuracy_score with the XGBoost model. And 0.855 for this RNN LSTM model. The accuracy of the two models is close. But compared to the XGBoost model, this RNN model takes less time to train. In the file model.py, we can also see that it's a simple model with just two layers. I think that it's not difficult to improve the performance of the model by changing some parameters.

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].

```
In [35]: # 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)
test_data, test_review_len = convert_and_pad(word_dict, test_review_words)
test_data = [np.array(test_data)]
```

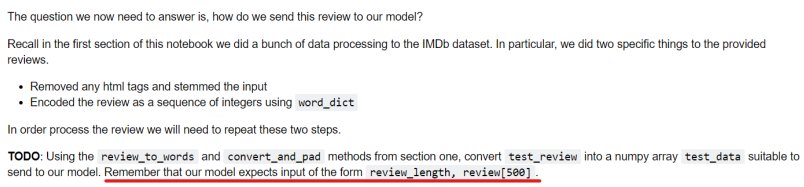
Now that we have processed the review, we can send the resulting array to our model to predict the sentiment of the review.

```
In [36]: predictor.predict(test_data)
Out[36]: array(0.5916678, 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(tes
t_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 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://aw7z0dc7z7.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.

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

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

Answer: I copied a 10/10 review from IMDB for the movie Inception, obviously it's a positive review. I got the right answer. I also tried a negative review and I got negative.

💡 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.

重新提交项目

下载项目

2 代码审阅评注 ›

了解 修改和重新提交项目的最佳做法.

返回 PATH