UDACITY

‹ Return to Classroom

Deploying a Sentiment Analysis Model

REVIEW

CODE REVIEW 3

HISTORY

▸ website/index.html 1

▸ train/train.py 1

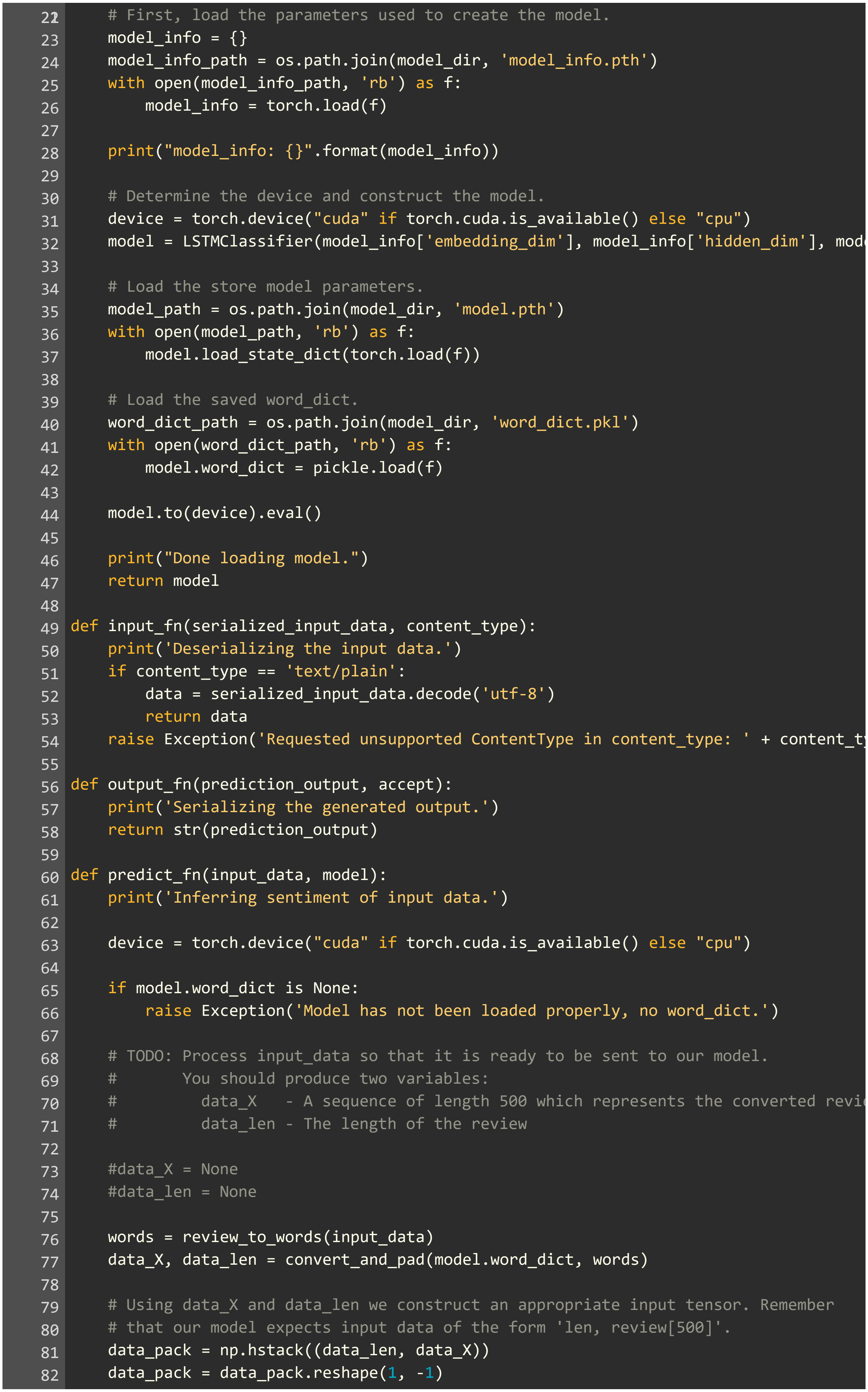
▾ serve/predict.py 1

```
import argparse
import json
import os
import pickle
import sys
import sagemaker_containers
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.utils.data

from model import LSTMClassifier

from utils import review_to_words, convert_and_pad

def model_fn(model_dir):
    """Load the PyTorch model from the `model_dir` directory."""
    print("Loading model.")
```



```
    # First, load the parameters used to create the model.
    model_info = {}
    model_info_path = os.path.join(model_dir, 'model_info.pth')
    with open(model_info_path, 'rb') as f:
        model_info = torch.load(f)

    print("model_info: {}".format(model_info))

    # Determine the device and construct the model.
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = LSTMClassifier(model_info['embedding_dim'], model_info['hidden_dim'], mode

    # Load the store model parameters.
    model_path = os.path.join(model_dir, 'model.pth')
    with open(model_path, 'rb') as f:
        model.load_state_dict(torch.load(f))

    # Load the saved word_dict.
    word_dict_path = os.path.join(model_dir, 'word_dict.pkl')
    with open(word_dict_path, 'rb') as f:
        model.word_dict = pickle.load(f)

    model.to(device).eval()

    print("Done loading model.")
    return model

def input_fn(serialized_input_data, content_type):
    print('Deserializing the input data.')
    if content_type == 'text/plain':
        data = serialized_input_data.decode('utf-8')
        return data
    raise Exception('Requested unsupported ContentType in content_type: ' + content_t

def output_fn(prediction_output, accept):
    print('Serializing the generated output.')
    return str(prediction_output)

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 revie
    #         data_len - The length of the review

    #data_X = None
    #data_len = None

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

    # 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))
    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
    #       be a numpy array which contains a single integer which is either 1 or 0

    with torch.no_grad():
        out = model.forward(data)

    result = np.rint(out.numpy())
```

AWESOME

Expressions correct

```


    return result

```

RETURN TO PATH