Plagiarism Detection Model ¶

Now that you've created training and test data, you are ready to define and train a model. Your goal in this notebook, will be to train a binary classification model that learns to label an answer file as either plagiarized or not, based on the features you provide the model.

This task will be broken down into a few discrete steps:

- Upload your data to S3.
- Define a binary classification model and a training script.
- Train your model and deploy it.
- Evaluate your deployed classifier and answer some questions about your approach.

To complete this notebook, you'll have to complete all given exercises and answer all the questions in this notebook.

> All your tasks will be clearly labeled **EXERCISE** and questions as **QUESTION**.

It will be up to you to explore different classification models and decide on a model that gives you the best performance for this dataset.

Load Data to S3

In the last notebook, you should have created two files: a `training.csv` and `test.csv` file with the features and class labels for the given corpus of plagiarized/non-plagiarized text data.

> The below cells load in some AWS SageMaker libraries and creates a default bucket. After creating this bucket, you can upload your locally stored data to S3.

Save your train and test `.csv` feature files, locally. To do this you can run the second notebook "2_Plagiarism_Feature_Engineering" in SageMaker or you can manually upload your files to this notebook using the upload icon in Jupyter Lab. Then you can upload local files to S3 by using `sagemaker_session.upload_data` and pointing directly to where the training data is saved.

In [1]:
```
import pandas as pd
import boto3
import sagemaker
```

In [2]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# session and role
sagemaker_session = sagemaker.Session()
role = sagemaker.get_execution_role()

# create an S3 bucket
bucket = sagemaker_session.default_bucket()
```

EXERCISE: Upload your training data to S3

Specify the `data_dir` where you've saved your `train.csv` file. Decide on a descriptive `prefix` that defines where your data will be uploaded in the default S3 bucket. Finally, create a pointer to your training data by calling `sagemaker_session.upload_data` and passing in the required parameters. It may help to look at the Session documentation (https://sagemaker.readthedocs.io/en/stable/session.html#sagemaker.session.Session.upload_data) or previous SageMaker code examples.

You are expected to upload your entire directory. Later, the training script will only access the `train.csv` file.

In [3]:
```
# should be the name of directory you created to save your features data
data_dir = 'plagiarism_data'

# set prefix, a descriptive name for a directory
prefix = 'project_Plagiarism_Detector'

# upload all data to S3
input_data = sagemaker_session.upload_data(path=data_dir, bucket=bucket, key_p
refix=prefix)
print(input_data)
```

```
s3://sagemaker-us-east-1-755811553671/project_Plagiarism_Detector
```

Test cell

Test that your data has been successfully uploaded. The below cell prints out the items in your S3 bucket and will throw an error if it is empty. You should see the contents of your `data_dir` and perhaps some checkpoints. If you see any other files listed, then you may have some old model files that you can delete via the S3 console (though, additional files shouldn't affect the performance of model developed in this notebook).

In [4]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# confirm that data is in S3 bucket
empty_check = []
for obj in boto3.resource('s3').Bucket(bucket).objects.all():
    empty_check.append(obj.key)
    print(obj.key)

assert len(empty_check) !=0, 'S3 bucket is empty.'
print('Test passed!')
```

```
project_Plagiarism_Detector/test.csv
project_Plagiarism_Detector/train.csv
Test passed!
```

Modeling

Now that you've uploaded your training data, it's time to define and train a model!

The type of model you create is up to you. For a binary classification task, you can choose to go one of three routes:

- Use a built-in classification algorithm, like LinearLearner.
- Define a custom Scikit-learn classifier, a comparison of models can be found here (https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html).
- Define a custom PyTorch neural network classifier.

It will be up to you to test out a variety of models and choose the best one. Your project will be graded on the accuracy of your final model.

EXERCISE: Complete a training script

To implement a custom classifier, you'll need to complete a `train.py` script. You've been given the folders `source_sklearn` and `source_pytorch` which hold starting code for a custom Scikit-learn model and a PyTorch model, respectively. Each directory has a `train.py` training script. To complete this project **you only need to complete one of these scripts**; the script that is responsible for training your final model.

A typical training script:

- Loads training data from a specified directory
- Parses any training & model hyperparameters (ex. nodes in a neural network, training epochs, etc.)
- Instantiates a model of your design, with any specified hyperparams
- Trains that model
- Finally, saves the model so that it can be hosted/deployed, later

Defining and training a model

Much of the training script code is provided for you. Almost all of your work will be done in the `if __name__ == '__main__':` section. To complete a `train.py` file, you will:

1. Import any extra libraries you need
2. Define any additional model training hyperparameters using `parser.add_argument`
3. Define a model in the `if __name__ == '__main__':` section
4. Train the model in that same section

Below, you can use `!pygmentize` to display an existing `train.py` file. Read through the code; all of your tasks are marked with `TODO` comments.

Note: If you choose to create a custom PyTorch model, you will be responsible for defining the model in the `model.py` file, and a `predict.py` file is provided. If you choose to use Scikit-learn, you only need a `train.py` file; you may import a classifier from the `sklearn` library.

In [5]:
```
# directory can be changed to: source_sklearn or source_pytorch
!pygmentize source_pytorch/train.py
```

```
import argparse
import json
import os
import pandas as pd
import torch
import torch.optim as optim
import torch.utils.data

# imports the model in model.py by name
from model import BinaryClassifier

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 = BinaryClassifier(model_info['input_features'], model_info['hidden
_dim'], model_info['output_dim'])

    # Load the stored 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))

    # set to eval mode, could use no_grad
    model.to(device).eval()

    print("Done loading model.")
    return model

# Gets training data in batches from the train.csv file
def _get_train_data_loader(batch_size, training_dir):
    print("Get train data loader.")

    train_data = pd.read_csv(os.path.join(training_dir, "train.csv"), header=
None, names=None)

    train_y = torch.from_numpy(train_data[[0]].values).float().squeeze()
    train_x = torch.from_numpy(train_data.drop([0], axis=1).values).float()

    train_ds = torch.utils.data.TensorDataset(train_x, train_y)

    return torch.utils.data.DataLoader(train_ds, batch_size=batch_size)


# Provided training function
def train(model, train_loader, epochs, criterion, optimizer, device):
    """
```

```
    This is the training method that is called by the PyTorch training scrip
t. The parameters
    passed are as follows:
    model        - The PyTorch model that we wish to train.
    train_loader - The PyTorch DataLoader that should be used during trainin
g.
    epochs       - The total number of epochs to train for.
    criterion    - The loss function used for training.
    optimizer    - The optimizer to use during training.
    device       - Where the model and data should be loaded (gpu or cpu).
    """

    # training loop is provided
    for epoch in range(1, epochs + 1):
        model.train() # Make sure that the model is in training mode.

        total_loss = 0

        for batch in train_loader:
            # get data
            batch_x, batch_y = batch

            batch_x = batch_x.to(device)
            batch_y = batch_y.to(device)

            optimizer.zero_grad()

            # get predictions from model
            y_pred = model(batch_x)

            # perform backprop
            loss = criterion(y_pred, batch_y)
            loss.backward()
            optimizer.step()

            total_loss += loss.data.item()

        print("Epoch: {}, Loss: {}".format(epoch, total_loss / len(train_load
er)))


## TODO: Complete the main code
if __name__ == '__main__':

    # All of the model parameters and training parameters are sent as argumen
ts
    # when this script is executed, during a training job

    # Here we set up an argument parser to easily access the parameters
    parser = argparse.ArgumentParser()

    # SageMaker parameters, like the directories for training data and saving
 models; set automatically
    # Do not need to change
    parser.add_argument('--output-data-dir', type=str, default=os.environ['SM
_OUTPUT_DATA_DIR'])
    parser.add_argument('--model-dir', type=str, default=os.environ['SM_MODEL
```

```
_DIR'])
    parser.add_argument('--data-dir', type=str, default=os.environ['SM_CHANNE
L_TRAIN'])

    # Training Parameters, given
    parser.add_argument('--batch-size', type=int, default=10, metavar='N',
                        help='input batch size for training (default: 10)')
    parser.add_argument('--epochs', type=int, default=10, metavar='N',
                        help='number of epochs to train (default: 10)')
    parser.add_argument('--seed', type=int, default=1, metavar='S',
                        help='random seed (default: 1)')

    ## TODO: Add args for the three model parameters: input_features, hidden_
dim, output_dim
    # Model Parameters
    parser.add_argument('--input_features', type=int, default=3, metavar='IN'
,
                        help='number of input features (default: 3)')
    parser.add_argument('--hidden_dim', type=int, default=100, metavar='H',
                        help='size of hidden dimension (default: 100)')
    parser.add_argument('--output_dim', type=int, default=1, metavar='OUT',
                        help='size of output dimension (default: 1)')

    # args holds all passed-in arguments
    args = parser.parse_args()

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    print("Using device {}.".format(device))

    torch.manual_seed(args.seed)

    # Load the training data.
    train_loader = _get_train_data_loader(args.batch_size, args.data_dir)


    ## --- Your code here --- ##

    ## TODO:  Build the model by passing in the input params
    # To get params from the parser, call args.argument_name, ex. args.epochs
or ards.hidden_dim
    # Don't forget to move your model .to(device) to move to GPU , if appropr
iate
    model = BinaryClassifier(args.input_features, args.hidden_dim, args.outpu
t_dim).to(device)

    ## TODO: Define an optimizer and loss function for training
    optimizer = optim.Adam(model.parameters(), lr=0.001, betas=(0.9, 0.999))
    criterion = torch.nn.BCELoss()

    # Trains the model (given line of code, which calls the above training fu
nction)
    train(model, train_loader, args.epochs, criterion, optimizer, device)

    ## TODO: complete in the model_info by adding three argument names, the f
irst is given
    # Keep the keys of this dictionary as they are
    model_info_path = os.path.join(args.model_dir, 'model_info.pth')
```

```
    with open(model_info_path, 'wb') as f:
        model_info = {
            'input_features': args.input_features,
            'hidden_dim': args.hidden_dim,
            'output_dim': args.output_dim,
        }
        torch.save(model_info, f)

    ## --- End of your code  --- ##


        # Save the model parameters
    model_path = os.path.join(args.model_dir, 'model.pth')
    with open(model_path, 'wb') as f:
        torch.save(model.cpu().state_dict(), f)
```

Provided code

If you read the code above, you can see that the starter code includes a few things:

- Model loading (`model_fn`) and saving code
- Getting SageMaker's default hyperparameters
- Loading the training data by name, `train.csv` and extracting the features and labels, `train_x` , and `train_y`

If you'd like to read more about model saving with joblib for sklearn (https://scikit-learn.org/stable/modules/model_persistence.html) or with torch.save (https://pytorch.org/tutorials/beginner/saving_loading_models.html), click on the provided links.

Create an Estimator

When a custom model is constructed in SageMaker, an entry point must be specified. This is the Python file which will be executed when the model is trained; the `train.py` function you specified above. To run a custom training script in SageMaker, construct an estimator, and fill in the appropriate constructor arguments:

- **entry_point**: The path to the Python script SageMaker runs for training and prediction.
- **source_dir**: The path to the training script directory `source_sklearn` OR `source_pytorch`.
- **entry_point**: The path to the Python script SageMaker runs for training and prediction.
- **source_dir**: The path to the training script directory `train_sklearn` OR `train_pytorch`.
- **entry_point**: The path to the Python script SageMaker runs for training.
- **source_dir**: The path to the training script directory `train_sklearn` OR `train_pytorch`.
- **role**: Role ARN, which was specified, above.
- **train_instance_count**: The number of training instances (should be left at 1).
- **train_instance_type**: The type of SageMaker instance for training. Note: Because Scikit-learn does not natively support GPU training, Sagemaker Scikit-learn does not currently support training on GPU instance types.
- **sagemaker_session**: The session used to train on Sagemaker.
- **hyperparameters** (optional): A dictionary `{'name':value, ..}` passed to the train function as hyperparameters.

Note: For a PyTorch model, there is another optional argument **framework_version**, which you can set to the latest version of PyTorch, `1.0`.

EXERCISE: Define a Scikit-learn or PyTorch estimator

To import your desired estimator, use one of the following lines:

```
from sagemaker.sklearn.estimator import SKLearn

from sagemaker.pytorch import PyTorch
```

In [6]:
```
# your import and estimator code, here
from sagemaker.pytorch import PyTorch

estimator = PyTorch(entry_point="train.py",
                    source_dir="source_pytorch",
                    role=role,
                    framework_version='1.0.0',
                    train_instance_count=1,
                    train_instance_type='ml.c4.xlarge',
                    hyperparameters={
                        'epochs': 80,
                        'hidden_dim': 100,
                        'input_features':3
                    })
```

EXERCISE: Train the estimator

Train your estimator on the training data stored in S3. This should create a training job that you can monitor in your SageMaker console.

In [7]:
```
%%time

# Train your estimator on S3 training data
estimator.fit({'train': input_data})
```

```
2020-01-16 06:11:42 Starting - Starting the training job...
2020-01-16 06:11:44 Starting - Launching requested ML instances......
2020-01-16 06:12:52 Starting - Preparing the instances for training......
2020-01-16 06:14:08 Downloading - Downloading input data
2020-01-16 06:14:08 Training - Downloading the training image..bash: cannot s
et terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
2020-01-16 06:14:22,506 sagemaker-containers INFO     Imported framework sage
maker_pytorch_container.training
2020-01-16 06:14:22,511 sagemaker-containers INFO     No GPUs detected (norma
l if no gpus installed)
2020-01-16 06:14:22,524 sagemaker_pytorch_container.training INFO     Block u
ntil all host DNS lookups succeed.
2020-01-16 06:14:22,525 sagemaker_pytorch_container.training INFO     Invokin
g user training script.
2020-01-16 06:14:22,787 sagemaker-containers INFO     Module train does not p
rovide a setup.py.
Generating setup.py
2020-01-16 06:14:22,787 sagemaker-containers INFO     Generating setup.cfg
2020-01-16 06:14:22,787 sagemaker-containers INFO     Generating MANIFEST.in
2020-01-16 06:14:22,788 sagemaker-containers INFO     Installing module with
 the following command:
/usr/bin/python -m pip install -U . -r requirements.txt
Processing /opt/ml/code
Collecting pandas (from -r requirements.txt (line 1))
  Downloading https://files.pythonhosted.org/packages/52/3f/f6a428599e0d4497e
1595030965b5ba455fd8ade6e977e3c819973c4b41d/pandas-0.25.3-cp36-cp36m-manylinu
x1_x86_64.whl (10.4MB)
Requirement already satisfied, skipping upgrade: python-dateutil>=2.6.1 in /u
sr/local/lib/python3.6/dist-packages (from pandas->-r requirements.txt (line
 1)) (2.8.0)
Requirement already satisfied, skipping upgrade: numpy>=1.13.3 in /usr/local/
lib/python3.6/dist-packages (from pandas->-r requirements.txt (line 1)) (1.1
6.4)
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /usr/local/l
ib/python3.6/dist-packages (from pandas->-r requirements.txt (line 1)) (2019.
1)
Requirement already satisfied, skipping upgrade: six>=1.5 in /usr/local/lib/p
ython3.6/dist-packages (from python-dateutil>=2.6.1->pandas->-r requirements.
txt (line 1)) (1.12.0)
Building wheels for collected packages: train
  Running setup.py bdist_wheel for train: started
  Running setup.py bdist_wheel for train: finished with status 'done'
  Stored in directory: /tmp/pip-ephem-wheel-cache-h8eiaip3/wheels/35/24/16/37
574d11bf9bde50616c67372a334f94fa8356bc7164af8ca3
Successfully built train
Installing collected packages: pandas, train
  Found existing installation: pandas 0.24.2
    Uninstalling pandas-0.24.2:
      Successfully uninstalled pandas-0.24.2

2020-01-16 06:14:37 Uploading - Uploading generated training modelSuccessfull
y installed pandas-0.25.3 train-1.0.0
You are using pip version 18.1, however version 19.3.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
2020-01-16 06:14:29,727 sagemaker-containers INFO     No GPUs detected (norma
l if no gpus installed)
```

```
2020-01-16 06:14:29,739 sagemaker-containers INFO     Invoking user script

Training Env:

{
    "additional_framework_parameters": {},
    "channel_input_dirs": {
        "train": "/opt/ml/input/data/train"
    },
    "current_host": "algo-1",
    "framework_module": "sagemaker_pytorch_container.training:main",
    "hosts": [
        "algo-1"
    ],
    "hyperparameters": {
        "hidden_dim": 100,
        "input_features": 3,
        "epochs": 80
    },
    "input_config_dir": "/opt/ml/input/config",
    "input_data_config": {
        "train": {
            "TrainingInputMode": "File",
            "S3DistributionType": "FullyReplicated",
            "RecordWrapperType": "None"
        }
    },
    "input_dir": "/opt/ml/input",
    "is_master": true,
    "job_name": "sagemaker-pytorch-2020-01-16-06-11-41-962",
    "log_level": 20,
    "master_hostname": "algo-1",
    "model_dir": "/opt/ml/model",
    "module_dir": "s3://sagemaker-us-east-1-755811553671/sagemaker-pytorch-20
20-01-16-06-11-41-962/source/sourcedir.tar.gz",
    "module_name": "train",
    "network_interface_name": "eth0",
    "num_cpus": 4,
    "num_gpus": 0,
    "output_data_dir": "/opt/ml/output/data",
    "output_dir": "/opt/ml/output",
    "output_intermediate_dir": "/opt/ml/output/intermediate",
    "resource_config": {
        "current_host": "algo-1",
        "hosts": [
            "algo-1"
        ],
        "network_interface_name": "eth0"
    },
    "user_entry_point": "train.py"
}

Environment variables:

SM_HOSTS=["algo-1"]
SM_NETWORK_INTERFACE_NAME=eth0
SM_HPS={"epochs":80,"hidden_dim":100,"input_features":3}
```

```
SM_USER_ENTRY_POINT=train.py
SM_FRAMEWORK_PARAMS={}
SM_RESOURCE_CONFIG={"current_host":"algo-1","hosts":["algo-1"],"network_inter
face_name":"eth0"}
SM_INPUT_DATA_CONFIG={"train":{"RecordWrapperType":"None","S3DistributionTyp
e":"FullyReplicated","TrainingInputMode":"File"}}
SM_OUTPUT_DATA_DIR=/opt/ml/output/data
SM_CHANNELS=["train"]
SM_CURRENT_HOST=algo-1
SM_MODULE_NAME=train
SM_LOG_LEVEL=20
SM_FRAMEWORK_MODULE=sagemaker_pytorch_container.training:main
SM_INPUT_DIR=/opt/ml/input
SM_INPUT_CONFIG_DIR=/opt/ml/input/config
SM_OUTPUT_DIR=/opt/ml/output
SM_NUM_CPUS=4
SM_NUM_GPUS=0
SM_MODEL_DIR=/opt/ml/model
SM_MODULE_DIR=s3://sagemaker-us-east-1-755811553671/sagemaker-pytorch-2020-01
-16-06-11-41-962/source/sourcedir.tar.gz
SM_TRAINING_ENV={"additional_framework_parameters":{},"channel_input_dirs":
{"train":"/opt/ml/input/data/train"},"current_host":"algo-1","framework_modul
e":"sagemaker_pytorch_container.training:main","hosts":["algo-1"],"hyperparam
eters":{"epochs":80,"hidden_dim":100,"input_features":3},"input_config_di
r":"/opt/ml/input/config","input_data_config":{"train":{"RecordWrapperTyp
e":"None","S3DistributionType":"FullyReplicated","TrainingInputMode":"Fil
e"}},"input_dir":"/opt/ml/input","is_master":true,"job_name":"sagemaker-pytor
ch-2020-01-16-06-11-41-962","log_level":20,"master_hostname":"algo-1","model_
dir":"/opt/ml/model","module_dir":"s3://sagemaker-us-east-1-755811553671/sage
maker-pytorch-2020-01-16-06-11-41-962/source/sourcedir.tar.gz","module_nam
e":"train","network_interface_name":"eth0","num_cpus":4,"num_gpus":0,"output_
data_dir":"/opt/ml/output/data","output_dir":"/opt/ml/output","output_interme
diate_dir":"/opt/ml/output/intermediate","resource_config":{"current_host":"a
lgo-1","hosts":["algo-1"],"network_interface_name":"eth0"},"user_entry_poin
t":"train.py"}
SM_USER_ARGS=["--epochs","80","--hidden_dim","100","--input_features","3"]
SM_OUTPUT_INTERMEDIATE_DIR=/opt/ml/output/intermediate
SM_CHANNEL_TRAIN=/opt/ml/input/data/train
SM_HP_HIDDEN_DIM=100
SM_HP_INPUT_FEATURES=3
SM_HP_EPOCHS=80
PYTHONPATH=/usr/local/bin:/usr/lib/python36.zip:/usr/lib/python3.6:/usr/lib/p
ython3.6/lib-dynload:/usr/local/lib/python3.6/dist-packages:/usr/lib/python3/
dist-packages

Invoking script with the following command:

/usr/bin/python -m train --epochs 80 --hidden_dim 100 --input_features 3


Using device cpu.
Get train data loader.
Epoch: 1, Loss: 0.6689546193395343
Epoch: 2, Loss: 0.6343606199537005
Epoch: 3, Loss: 0.6036822540419442
Epoch: 4, Loss: 0.5786380299500057
Epoch: 5, Loss: 0.5519566663673946
```

```
Epoch: 6, Loss: 0.5145282575062343
Epoch: 7, Loss: 0.49306176389966694
Epoch: 8, Loss: 0.4458782374858856
Epoch: 9, Loss: 0.40573562894548687
Epoch: 10, Loss: 0.3807752771036966
Epoch: 11, Loss: 0.343296080827713
Epoch: 12, Loss: 0.3212815395423344
Epoch: 13, Loss: 0.3107548973390034
Epoch: 14, Loss: 0.2605103573628834
Epoch: 15, Loss: 0.29052311394895824
Epoch: 16, Loss: 0.2812582648226193
Epoch: 17, Loss: 0.2810427376202175
Epoch: 18, Loss: 0.2434398297752653
Epoch: 19, Loss: 0.23729423433542252
Epoch: 20, Loss: 0.24735025210039957
Epoch: 21, Loss: 0.2887136595589774
Epoch: 22, Loss: 0.24430437705346517
Epoch: 23, Loss: 0.2433713557464736
Epoch: 24, Loss: 0.2474848979285785
Epoch: 25, Loss: 0.24240509420633316
Epoch: 26, Loss: 0.2547099579657827
Epoch: 27, Loss: 0.22930930129119328
Epoch: 28, Loss: 0.2607113628515175
Epoch: 29, Loss: 0.22197170395936286
Epoch: 30, Loss: 0.2459846713713237
Epoch: 31, Loss: 0.23087905134473527
Epoch: 32, Loss: 0.25018177926540375
Epoch: 33, Loss: 0.24978476709553174
Epoch: 34, Loss: 0.26882472687533926
Epoch: 35, Loss: 0.2190635640706335
Epoch: 36, Loss: 0.2276241279074124
Epoch: 37, Loss: 0.2191994967205184
Epoch: 38, Loss: 0.22473067683832987
Epoch: 39, Loss: 0.20065285904066904
Epoch: 40, Loss: 0.21415736100503377
Epoch: 41, Loss: 0.21611637994647026
Epoch: 42, Loss: 0.20469064159052713
Epoch: 43, Loss: 0.24586070222514017
Epoch: 44, Loss: 0.22964913185153688
Epoch: 45, Loss: 0.23549561042870795
Epoch: 46, Loss: 0.2223965690604278
Epoch: 47, Loss: 0.2392499723604747
Epoch: 48, Loss: 0.23229094807590758
Epoch: 49, Loss: 0.23724826904279844
Epoch: 50, Loss: 0.22410243909273828
Epoch: 51, Loss: 0.2152696331696851
Epoch: 52, Loss: 0.22064315580895968
Epoch: 53, Loss: 0.21581963396498136
Epoch: 54, Loss: 0.23024961884532655
Epoch: 55, Loss: 0.20389250133718764
Epoch: 56, Loss: 0.2124295011162758
Epoch: 57, Loss: 0.2164360364632947
Epoch: 58, Loss: 0.21767410529511316
Epoch: 59, Loss: 0.25163458394152777
Epoch: 60, Loss: 0.19800245336123876
Epoch: 61, Loss: 0.20307296620947973
Epoch: 62, Loss: 0.21221275787268365
```

```
Epoch: 63, Loss: 0.22698731241481646
Epoch: 64, Loss: 0.20812575731958663
Epoch: 65, Loss: 0.2413950456040246
Epoch: 66, Loss: 0.2284870823579175
Epoch: 67, Loss: 0.2359508057790143
Epoch: 68, Loss: 0.21875562518835068
Epoch: 69, Loss: 0.20409264362284116
Epoch: 70, Loss: 0.2034741065331868
Epoch: 71, Loss: 0.22669539440955436
Epoch: 72, Loss: 0.235386780330113
Epoch: 73, Loss: 0.21137151654277528
Epoch: 74, Loss: 0.22285349693681514
Epoch: 75, Loss: 0.2059735075703689
Epoch: 76, Loss: 0.22766907513141632
Epoch: 77, Loss: 0.21026731069598878
Epoch: 78, Loss: 0.23291317692824773
Epoch: 79, Loss: 0.22290450281330518
Epoch: 80, Loss: 0.1927881741097995
2020-01-16 06:14:32,122 sagemaker-containers INFO     Reporting training SUCC
ESS

2020-01-16 06:14:43 Completed - Training job completed
Training seconds: 57
Billable seconds: 57
CPU times: user 527 ms, sys: 23.5 ms, total: 550 ms
Wall time: 3min 42s
```

EXERCISE: Deploy the trained model

After training, deploy your model to create a `predictor`. If you're using a PyTorch model, you'll need to create a trained `PyTorchModel` that accepts the trained `<model>.model_data` as an input parameter and points to the provided `source_pytorch/predict.py` file as an entry point.

To deploy a trained model, you'll use `<model>.deploy`, which takes in two arguments:

- **initial_instance_count**: The number of deployed instances (1).
- **instance_type**: The type of SageMaker instance for deployment.

Note: If you run into an instance error, it may be because you chose the wrong training or deployment instance_type. It may help to refer to your previous exercise code to see which types of instances we used.

In [8]:
```
%%time

# uncomment, if needed
from sagemaker.pytorch import PyTorchModel


# deploy your model to create a predictor
predictor = estimator.deploy(initial_instance_count = 1, instance_type = 'ml.t
2.medium')
```

```
-------------------------------------------------------------------------------
----------------------------------------------!CPU times: user 621 ms, sys: 2
3.7 ms, total: 645 ms
Wall time: 10min 21s
```

Evaluating Your Model

Once your model is deployed, you can see how it performs when applied to our test data.

The provided cell below, reads in the test data, assuming it is stored locally in `data_dir` and named `test.csv`. The labels and features are extracted from the `.csv` file.

In [9]:
```
"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
import os

# read in test data, assuming it is stored locally
test_data = pd.read_csv(os.path.join(data_dir, "test.csv"), header=None, names
=None)

# labels are in the first column
test_y = test_data.iloc[:,0]
test_x = test_data.iloc[:,1:]
```

EXERCISE: Determine the accuracy of your model

Use your deployed `predictor` to generate predicted, class labels for the test data. Compare those to the *true* labels, `test_y`, and calculate the accuracy as a value between 0 and 1.0 that indicates the fraction of test data that your model classified correctly. You may use sklearn.metrics (https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics) for this calculation.

To pass this project, your model should get at least 90% test accuracy.

In [10]:
```
# First: generate predicted, class labels
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

test_y_preds = predictor.predict(torch.from_numpy(test_x.values).float().to(de
vice))



"""
DON'T MODIFY ANYTHING IN THIS CELL THAT IS BELOW THIS LINE
"""
# test that your model generates the correct number of labels
assert len(test_y_preds)==len(test_y), 'Unexpected number of predictions.'
print('Test passed!')
```

```
Test passed!
```

In [11]:
```
import numpy as np
# code to evaluate the endpoint on test data
# returns a variety of model metrics
def evaluate(test_y_preds, test_labels, verbose=True):
    """
    Evaluate a model on a test set given the prediction endpoint.
    Return binary classification metrics.
    :param predictor: A prediction endpoint
    :param test_features: Test features
    :param test_labels: Class labels for test data
    :param verbose: If True, prints a table of all performance metrics
    :return: A dictionary of performance metrics.
    """

    # rounding and squeezing array
    test_preds = pd.Series(np.squeeze(np.round(test_y_preds).astype(int)))

    # calculate true positives, false positives, true negatives, false negativ
es
    tp = np.logical_and(test_labels, test_preds).sum()
    fp = np.logical_and(1-test_labels, test_preds).sum()
    tn = np.logical_and(1-test_labels, 1-test_preds).sum()
    fn = np.logical_and(test_labels, 1-test_preds).sum()

    # calculate binary classification metrics
    recall = tp / (tp + fn)
    precision = tp / (tp + fp)
    accuracy = (tp + tn) / (tp + fp + tn + fn)

    # print metrics
    if verbose:
        print(pd.crosstab(test_labels, test_preds, rownames=['actuals'], colna
mes=['predictions']))
        print("\n{:<11} {:.3f}".format('Recall:', recall))
        print("{:<11} {:.3f}".format('Precision:', precision))
        print("{:<11} {:.3f}".format('Accuracy:', accuracy))
        print()

    return {'TP': tp, 'FP': fp, 'FN': fn, 'TN': tn,
            'Precision': precision, 'Recall': recall, 'Accuracy': accuracy}
```

In [12]:
```
# get metrics for custom predictor
metrics = evaluate(test_y_preds, test_y, True)
```

```
predictions   0   1
actuals
0            10   0
1             0  15

Recall:     1.000
Precision:  1.000
Accuracy:   1.000
```

In [13]:
```
print(metrics)
```

```
{'TP': 15, 'FP': 0, 'FN': 0, 'TN': 10, 'Precision': 1.0, 'Recall': 1.0, 'Accu
racy': 1.0}
```

In [14]:
```
# Second: calculate the test accuracy

accuracy = metrics['Accuracy']

print(accuracy)


## print out the array of predicted and true labels, if you want
print('\nPredicted class labels: ')
print(np.squeeze(np.round(test_y_preds).astype(int)))
print('\nTrue class labels: ')
print(test_y.values)
```

```
1.0

Predicted class labels:
[1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0]

True class labels:
[1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 1 0 1 1 0 0]
```

Question 1: How many false positives and false negatives did your model produce, if any? And why do you think this is?

Answer: False positive = 0 , False negative = 0. The reason for this is neural network was able to identify appropriate weights and bias to identify pattern in data such that it was linearly separable. This is primarily due to small dataset and selection of containment features having lowest correlation the highest lcs_norm. With large dataset, it is possible that we have lower accuracy overall.

Question 2: How did you decide on the type of model to use?

Answer: Based on the binary classification and complexity of the problem, I chose simple linear neural network with 2 fully connected layers. With little tweaking of epochs(80) and hidden dimension (100), I was able to bring down loss function to 0.19 and 100% accuracy.

EXERCISE: Clean up Resources

After you're done evaluating your model, **delete your model endpoint**. You can do this with a call to `.delete_endpoint()` . You need to show, in this notebook, that the endpoint was deleted. Any other resources, you may delete from the AWS console, and you will find more instructions on cleaning up all your resources, below.

In [15]:
```
# uncomment and fill in the line below!
# <name_of_deployed_predictor>.delete_endpoint()
predictor.delete_endpoint()
```

Deleting S3 bucket

When you are *completely* done with training and testing models, you can also delete your entire S3 bucket. If you do this before you are done training your model, you'll have to recreate your S3 bucket and upload your training data again.

In [16]:
```
# deleting bucket, uncomment lines below

bucket_to_delete = boto3.resource('s3').Bucket(bucket)
bucket_to_delete.objects.all().delete()
```

Out[16]:
```
[{'ResponseMetadata': {'RequestId': '22E7A2BF5087604B',
   'HostId': 'DB4coyt0+I1PMMZFg0fn/GvdcnGboqxOq3LUeGVd3A7JIdsPZ4QwmG2Qj5eqEHM
ev1R0HDBBAnQ=',
   'HTTPStatusCode': 200,
   'HTTPHeaders': {'x-amz-id-2': 'DB4coyt0+I1PMMZFg0fn/GvdcnGboqxOq3LUeGVd3A7
JIdsPZ4QwmG2Qj5eqEHMev1R0HDBBAnQ=',
    'x-amz-request-id': '22E7A2BF5087604B',
    'date': 'Thu, 16 Jan 2020 06:25:47 GMT',
    'connection': 'close',
    'content-type': 'application/xml',
    'transfer-encoding': 'chunked',
    'server': 'AmazonS3'},
   'RetryAttempts': 0},
  'Deleted': [{'Key': 'project_Plagiarism_Detector/train.csv'},
   {'Key': 'sagemaker-pytorch-2020-01-16-06-11-41-962/output/model.tar.gz'},
   {'Key': 'project_Plagiarism_Detector/test.csv'},
   {'Key': 'sagemaker-pytorch-2020-01-16-06-11-41-962/debug-output/training_j
ob_end.ts'},
   {'Key': 'sagemaker-pytorch-2020-01-16-06-11-41-962/source/sourcedir.tar.g
z'}]}]
```

Deleting all your models and instances

When you are *completely* done with this project and do **not** ever want to revisit this notebook, you can choose to delete all of your SageMaker notebook instances and models by following [these instructions (https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-cleanup.html)](https://docs.aws.amazon.com/sagemaker/latest/dg/ex1-cleanup.html). Before you delete this notebook instance, I recommend at least downloading a copy and saving it, locally.

Further Directions

There are many ways to improve or add on to this project to expand your learning or make this more of a unique project for you. A few ideas are listed below:

- Train a classifier to predict the *category* (1-3) of plagiarism and not just plagiarized (1) or not (0).
- Utilize a different and larger dataset to see if this model can be extended to other types of plagiarism.
- Use language or character-level analysis to find different (and more) similarity features.
- Write a complete pipeline function that accepts a source text and submitted text file, and classifies the submitted text as plagiarized or not.
- Use API Gateway and a lambda function to deploy your model to a web application.

These are all just options for extending your work. If you've completed all the exercises in this notebook, you've completed a real-world application, and can proceed to submit your project. Great job!