Optimize models using Automatic Model Tuning

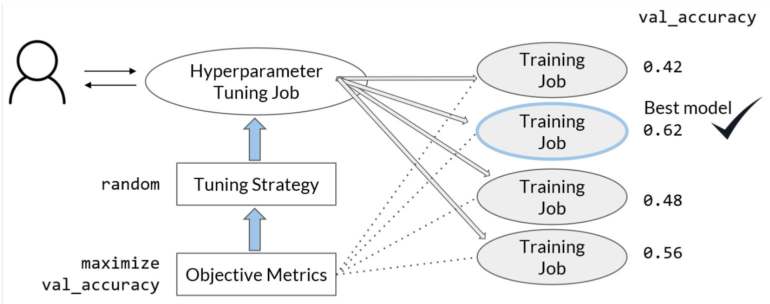
Introduction

When training ML models, hyperparameter tuning is a step taken to find the best performing training model. In this lab you will apply a random algorithm of Automated Hyperparameter Tuning to train a BERT-based natural language processing (NLP) classifier. The model analyzes customer feedback and classifies the messages into positive (1), neutral (0), and negative (-1) sentiments.

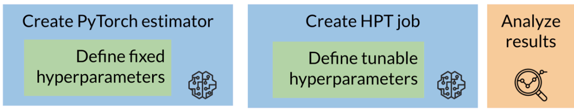
Table of Contents

Amazon SageMaker supports Automated Hyperparameter Tuning. It runs multiple training jobs on the training dataset using the hyperparameter ranges specified by the user. Then it chooses the combination of hyperparameters that leads to the best model candidate. The choice is made based on the objective metrics, e.g. maximization of the validation accuracy.

For the choice of hyperparameters combinations, SageMaker supports two different types of tuning strategies: random and Bayesian. This capability can be further extended by providing an implementation of a custom tuning strategy as a Docker container.



In this lab you will perform the following three steps:



First, let's install and import the required modules.

In [1]:
```
# please ignore warning messages during the installation
!pip install --disable-pip-version-check -q sagemaker==2.35.0
!conda install -q -y pytorch==1.6.0 -c pytorch
!pip install --disable-pip-version-check -q transformers==3.5.1
```

```
/opt/conda/lib/python3.7/site-packages/secretstorage/dhcrypto.py:16: Cryp
tographyDeprecationWarning: int_from_bytes is deprecated, use int.from_by
tes instead
  from cryptography.utils import int_from_bytes
/opt/conda/lib/python3.7/site-packages/secretstorage/util.py:25: Cryptogr
aphyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes
instead
  from cryptography.utils import int_from_bytes
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommen
ded to use a virtual environment instead: https://pip.pypa.io/warnings/ve
nv
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.

/opt/conda/lib/python3.7/site-packages/secretstorage/dhcrypto.py:16: Cryp
tographyDeprecationWarning: int_from_bytes is deprecated, use int.from_by
tes instead
  from cryptography.utils import int_from_bytes
/opt/conda/lib/python3.7/site-packages/secretstorage/util.py:25: Cryptogr
aphyDeprecationWarning: int_from_bytes is deprecated, use int.from_bytes
instead
  from cryptography.utils import int_from_bytes
WARNING: Running pip as the 'root' user can result in broken permissions
and conflicting behaviour with the system package manager. It is recommen
ded to use a virtual environment instead: https://pip.pypa.io/warnings/ve
nv
```

In [2]:
```
import boto3
import sagemaker
import pandas as pd
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c3/w1')

# low-level service client of the boto3 session
sm = boto3.client(service_name='sagemaker',
                  config=config)

sess = sagemaker.Session(sagemaker_client=sm)

bucket = sess.default_bucket()
role = sagemaker.get_execution_role()
region = sess.boto_region_name
```

1. Configure dataset and Hyperparameter Tuning Job (HTP)

1.1. Configure dataset

Let's set up the paths and copy the data to the S3 bucket:

In [3]:
```
processed_train_data_s3_uri = 's3://{}/transformed/data/sentiment-train/'
processed_validation_data_s3_uri = 's3://{}/transformed/data/sentiment-va
processed_test_data_s3_uri = 's3://{}/transformed/data/sentiment-test/'.f
```

Upload the data to the S3 bucket:

In [4]:
```
!aws s3 cp --recursive ./data/sentiment-train $processed_train_data_s3_ur
!aws s3 cp --recursive ./data/sentiment-validation $processed_validation_
!aws s3 cp --recursive ./data/sentiment-test $processed_test_data_s3_uri
```

```
upload: data/sentiment-train/part-algo-1-womens_clothing_ecommerce_review
s.tsv to s3://sagemaker-us-east-1-593402094095/transformed/data/sentiment
-train/part-algo-1-womens_clothing_ecommerce_reviews.tsv
upload: data/sentiment-validation/part-algo-1-womens_clothing_ecommerce_r
eviews.tsv to s3://sagemaker-us-east-1-593402094095/transformed/data/sent
iment-validation/part-algo-1-womens_clothing_ecommerce_reviews.tsv
upload: data/sentiment-test/part-algo-1-womens_clothing_ecommerce_reviews
.tsv to s3://sagemaker-us-east-1-593402094095/transformed/data/sentiment-
test/part-algo-1-womens_clothing_ecommerce_reviews.tsv
```

Check the existence of those files in the S3 bucket:

In [5]:
```
!aws s3 ls --recursive $processed_train_data_s3_uri
```

```
2021-09-16 08:57:37    4894416 transformed/data/sentiment-train/part-algo
-1-womens_clothing_ecommerce_reviews.tsv
```

In [6]:
```
!aws s3 ls --recursive $processed_validation_data_s3_uri
```

```
2021-09-16 08:57:37     276522 transformed/data/sentiment-validation/part-algo-1-womens_clothing_ecommerce_reviews.tsv
```

In [7]:
```
!aws s3 ls --recursive $processed_test_data_s3_uri
```

```
2021-09-16 08:57:38     273414 transformed/data/sentiment-test/part-algo-1-womens_clothing_ecommerce_reviews.tsv
```

Exercise 1

Set up a dictionary of the input training and validation data channels, wrapping the corresponding S3 locations in a `TrainingInput` object.

Instructions: Pass the S3 input paths for training and validation data into the `TrainingInput` function

```
TrainingInput(s3_data=...)
```

to construct the Amazon SageMaker channels for S3 input data sources. Then put the corresponding channels into the dictionary.

In [8]:
```
from sagemaker.inputs import TrainingInput

data_channels = {
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    'train': processed_train_data_s3_uri, # Replace None
    'validation': processed_validation_data_s3_uri # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
}
```

There is no need to create a test data channel, as the test data is used later at the evaluation stage and does not need to be wrapped into the `sagemaker.inputs.TrainingInput` function.

1.2. Configure Hyperparameter Tuning Job

Model hyperparameters need to be set prior to starting the model training as they control the process of learning. Some of the hyperparameters you will set up as static - they will not be explored during the tuning job. For the non-static hyperparameters you will set the range of possible values to be explored.

First, configure static hyperparameters including the instance type, instance count, maximum sequence length, etc. For the purposes of this lab, you will use a relatively small instance type. Please refer to this link for additional instance types that may work for your use cases outside of this lab.

In [9]:
```
max_seq_length=128 # maximum number of input tokens passed to BERT model
freeze_bert_layer=False # specifies the depth of training within the netw
epochs=3
train_steps_per_epoch=50
validation_batch_size=64
validation_steps_per_epoch=50
seed=42

train_instance_count=1
train_instance_type='ml.c5.9xlarge'
train_volume_size=256
input_mode='File'
run_validation=True
```

Some of these will be passed into the PyTorch estimator and tuner in the hyperparameters argument. Let's set up the dictionary for that:

In [10]:
```
hyperparameters_static={
    'freeze_bert_layer': freeze_bert_layer,
    'max_seq_length': max_seq_length,
    'epochs': epochs,
    'train_steps_per_epoch': train_steps_per_epoch,
    'validation_batch_size': validation_batch_size,
    'validation_steps_per_epoch': validation_steps_per_epoch,
    'seed': seed,
    'run_validation': run_validation
}
```

Configure hyperparameter ranges to explore in the Tuning Job. The values of the ranges typically come from prior experience, research papers, or other models similar to the task you are trying to do.

In [11]:
```
from sagemaker.tuner import IntegerParameter
from sagemaker.tuner import ContinuousParameter
from sagemaker.tuner import CategoricalParameter

hyperparameter_ranges = {
    'learning_rate': ContinuousParameter(0.00001, 0.00005, scaling_type='
    'train_batch_size': CategoricalParameter([128, 256]), # specifying ca
}
```

1.3. Set up evaluation metrics

Choose loss and accuracy as the evaluation metrics. The regular expressions `Regex` will capture the values of metrics that the algorithm will emit.

In [12]:
```
metric_definitions = [
     {'Name': 'validation:loss', 'Regex': 'val_loss: ([0-9.]+)'},
     {'Name': 'validation:accuracy', 'Regex': 'val_acc: ([0-9.]+)'},
]
```

For example, these sample log lines...

```
[step: 100] val_loss: 0.76 - val_acc: 70.92%
```

...will produce the following metrics in CloudWatch:

`validation:loss` = 0.76

`validation:accuracy` = 70.92

In the Tuning Job, you will be maximizing validation accuracy as the objective metric.

2. Run Tuning Job

2.1. Set up the RoBERTa and PyTorch script to run on SageMaker

Prepare the PyTorch model to run as a SageMaker Training Job. The estimator takes into the entry point a separate Python file, which will be called during the training. You can open and review this file src/train.py.

For more information on the `PyTorchEstimator`, see the documentation here: https://sagemaker.readthedocs.io/

In [13]:

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator
# Note: indeed, it is not compulsory to rename the PyTorch estimator,
# but this is useful for code clarity, especially when a few modules of '

estimator = PyTorchEstimator(
    entry_point='train.py',
    source_dir='src',
    role=role,
    instance_count=train_instance_count,
    instance_type=train_instance_type,
    volume_size=train_volume_size,
    py_version='py3',
    framework_version='1.6.0',
    hyperparameters=hyperparameters_static,
    metric_definitions=metric_definitions,
    input_mode=input_mode,
)
```

2.2. Launch the Hyperparameter Tuning Job

A hyperparameter tuning job runs a series of training jobs that each test a combination of hyperparameters for a given objective metric (i.e. `validation:accuracy`). In this lab, you will use a `Random` search strategy to determine the combinations of hyperparameters - within the specific ranges - to use for each training job within the tuning job. For more information on hyperparameter tuning search strategies, please see the following documentation: https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning-how-it-works.html

When the tuning job completes, you can select the hyperparameters used by the best-performing training job relative to the objective metric.

The `max_jobs` parameter is a stop criteria that limits the number of overall training jobs (and therefore hyperparameter combinations) to run within the tuning job.

The `max_parallel_jobs` parameter limits the number of training jobs (and therefore hyperparameter combinations) to run in parallel within the tuning job. This parameter is often used in combination with the `Bayesian` search strategy when you want to test a smaller set of training jobs (less than the `max_jobs`), learn from the smaller set of training jobs, then apply Bayesian methods to determine the next set of hyperparameters used by the next set of training jobs. Bayesian methods can improve hyperparameter-tuning performance in some cases.

The `early_stopping_type` parameter is used by SageMaker hyper-parameter tuning jobs to automatically stop a training job if the job is not improving the objective metrics (i.e. `validation:accuracy`) relative to previous training jobs within the tuning job. For more information on early stopping, please see the following documentation: https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning-early-stopping.html.

Exercise 2

Set up the Hyperparameter Tuner.

Instructions: Use the function `HyperparameterTuner`, passing the variables defined above. Please use tuning strategy `'Random'`.

```
tuner = HyperparameterTuner(
    estimator=..., # estimator
    hyperparameter_ranges=..., # hyperparameter ranges
    metric_definitions=..., # definition metric
    strategy='...', # tuning strategy
    objective_type='Maximize',
    objective_metric_name='validation:accuracy',
    max_jobs=2, # maximum number of jobs to run
    max_parallel_jobs=2, # maximum number of jobs to run in parallel
    early_stopping_type='Auto' # early stopping criteria
)
```

In [14]:
```
from sagemaker.tuner import HyperparameterTuner

tuner = HyperparameterTuner(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    estimator=estimator, # Replace None
    hyperparameter_ranges=hyperparameter_ranges, # Replace None
    metric_definitions=metric_definitions, # Replace None
    strategy='Random', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    objective_type='Maximize',
    objective_metric_name='validation:accuracy',
    max_jobs=2, # maximum number of jobs to run
    max_parallel_jobs=2, # maximum number of jobs to run in parallel
    early_stopping_type='Auto' # early stopping criteria
)
```

Exercise 3

Launch the SageMaker Hyper-Parameter Tuning (HPT) Job.

Instructions: Use the `tuner.fit` function, passing the configured train and validation inputs (data channels).

```
tuner.fit(
    inputs=..., # train and validation input
    include_cls_metadata=False, # to be set as false if the algorithm cannot handle unknown hyperparameters
    wait=False # do not wait for the job to complete before continuing
)
```

In [15]:
```
tuner.fit(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    inputs=data_channels, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    include_cls_metadata=False,
    wait=False
)
```

2.3. Check Tuning Job status

You can see the Tuning Job status in the console. Let's get the Tuning Job name to construct the link.

In [16]:
```
tuning_job_name = tuner.latest_tuning_job.job_name
print(tuning_job_name)
```

```
pytorch-training-210916-0857
```

Check the status of the Tuning Job.

In [17]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="blank" href="https://console.aws.amazo
```

Review **Hyper-Parameter Tuning Job**

Wait for the Tuning Job to complete.

This cell will take approximately 20-30 minutes to run.

In [18]:
```
%%time

tuner.wait()
```

```
........................................................................
........................................................................
........................................................................
........................................................................
..............................................................!
CPU times: user 1.38 s, sys: 226 ms, total: 1.6 s
Wall time: 29min 48s
```

Wait until the ^^ Tuning Job ^^ completes above

The results of the SageMaker Hyperparameter Tuning Job are available on the `analytics` of the `tuner object`. The `dataframe` function converts the result directly into the dataframe. You can explore the results with the following lines of the code:

In [19]:
```python
import time

time.sleep(10) # slight delay to allow the analytics to be calculated

df_results = tuner.analytics().dataframe()
df_results.shape
```

Out[19]:
```
(2, 8)
```

In [20]:
```python
df_results.sort_values('FinalObjectiveValue', ascending=0)
```

Out[20]:

	learning_rate	train_batch_size	TrainingJobName	TrainingJobStatus	FinalObjectiveVal
0	0.000020	"128"	pytorch-training-210916-0857-002-fd13714c	Completed	70.6999
1	0.000015	"128"	pytorch-training-210916-0857-001-fa4b6e1a	Stopped	37.1100

When training and tuning at scale, it is important to continuously monitor and use the right compute resources. While you have the flexibility of choosing different compute options how do you choose the specific instance types and sizes to use? There is no standard answer for this. It comes down to understanding the workload and running empirical testing to determine the best compute resources to use for the training.

SageMaker Training Jobs emit CloudWatch metrics for resource utilization. You can review them in the AWS console:

- open the link
- notice that you are in the section Amazon SageMaker -> Hyperparameter tuning jobs
- have a look at the list of the Training jobs below and click on one of them
- scroll down to the Monitor section and review the available metrics

In [21]:
```python
from IPython.core.display import display, HTML

display(HTML('<b>Review Training Jobs of the <a target="blank" href="http
```

Review Training Jobs of the Hyper-Parameter Tuning Job

3. Evaluate the results

An important part of developing a model is evaluating the model with a test data set - one that the model has never seen during its training process. The final metrics resulting from this evaluation can be used to compare competing machine learning models. The higher the value of these metrics, the better the model is able to generalize.

3.1. Show the best candidate

Exercise 4

Show the best candidate - the one with the highest accuracy result.

Instructions: Use the `sort_values` function to sort the results by accuracy, which is stored in the column `FinalObjectiveValue`. Put `ascending=0` and `head(1)` for the selection.

```
df_results.sort_values(
    '...', # column name for sorting
    ascending=0).head(1)
```

In [22]:
```
df_results.sort_values(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    'FinalObjectiveValue', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    ascending=0).head(1)
```

Out[22]:

	learning_rate	train_batch_size	TrainingJobName	TrainingJobStatus	FinalObjectiveVal
0	0.00002	"128"	pytorch-training-210916-0857-002-fd13714c	Completed	70.6999

3.2. Evaluate the best candidate

Let's pull the information about the best candidate from the dataframe and then take the Training Job name from the column `TrainingJobName`.

In [23]:
```
best_candidate = df_results.sort_values('FinalObjectiveValue', ascending=

best_candidate_training_job_name = best_candidate['TrainingJobName']
print('Best candidate Training Job name: {}'.format(best_candidate_traini
```

```
Best candidate Training Job name: pytorch-training-210916-0857-002-fd1371
4c
```

Exercise 5

Show accuracy result for the best candidate.

Instructions: Use the example in the cell above.

In [24]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
best_candidate_accuracy = best_candidate['FinalObjectiveValue'] # Replace
### END SOLUTION - DO NOT delete this comment for grading purposes

print('Best candidate accuracy result: {}'.format(best_candidate_accuracy
```

```
Best candidate accuracy result: 70.69999694824219
```

You can use the function `describe_training_job` of the service client to get some more information about the best candidate. The result is in dictionary format. Let's check that it has the same Training Job name:

In [25]:
```
best_candidate_description = sm.describe_training_job(TrainingJobName=bes

best_candidate_training_job_name2 = best_candidate_description['TrainingJ

print('Training Job name: {}'.format(best_candidate_training_job_name2))
```

```
Training Job name: pytorch-training-210916-0857-002-fd13714c
```

Exercise 6

Pull the Tuning Job and Training Job Amazon Resource Name (ARN) from the best candidate training job description.

Instructions: Print the keys of the best candidate Training Job description dictionary, choose the ones related to the Tuning Job and Training Job ARN and print their values.

In [26]:
```
print(best_candidate_description.keys())
```

```
dict_keys(['TrainingJobName', 'TrainingJobArn', 'TuningJobArn', 'ModelArt
ifacts', 'TrainingJobStatus', 'SecondaryStatus', 'HyperParameters', 'Algo
rithmSpecification', 'RoleArn', 'InputDataConfig', 'OutputDataConfig', 'R
esourceConfig', 'StoppingCondition', 'CreationTime', 'TrainingStartTime',
'TrainingEndTime', 'LastModifiedTime', 'SecondaryStatusTransitions', 'Fin
alMetricDataList', 'EnableNetworkIsolation', 'EnableInterContainerTraffic
Encryption', 'EnableManagedSpotTraining', 'TrainingTimeInSeconds', 'Billa
bleTimeInSeconds', 'ProfilingStatus', 'ResponseMetadata'])
```

In [27]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
best_candidate_tuning_job_arn = best_candidate_description['TuningJobArn'
best_candidate_training_job_arn = best_candidate_description['TrainingJob
### END SOLUTION - DO NOT delete this comment for grading purposes
print('Best candidate Tuning Job ARN: {}'.format(best_candidate_tuning_jo
print('Best candidate Training Job ARN: {}'.format(best_candidate_trainin
```

```
Best candidate Tuning Job ARN: arn:aws:sagemaker:us-east-1:593402094095:h
yper-parameter-tuning-job/pytorch-training-210916-0857
Best candidate Training Job ARN: arn:aws:sagemaker:us-east-1:593402094095
:training-job/pytorch-training-210916-0857-002-fd13714c
```

Pull the path of the best candidate model in the S3 bucket. You will need it later to set up the Processing Job for the evaluation.

In [28]:
```
model_tar_s3_uri = sm.describe_training_job(TrainingJobName=best_candidat
print(model_tar_s3_uri)
```

```
s3://sagemaker-us-east-1-593402094095/pytorch-training-210916-0857-002-fd
13714c/output/model.tar.gz
```

To perform model evaluation you will use a scikit-learn-based Processing Job. This is essentially a generic Python Processing Job with scikit-learn pre-installed. You can specify the version of scikit-learn you wish to use. Also pass the SageMaker execution role, processing instance type and instance count.

In [29]:
```
from sagemaker.sklearn.processing import SKLearnProcessor

processing_instance_type = "ml.c5.2xlarge"
processing_instance_count = 1

processor = SKLearnProcessor(
    framework_version="0.23-1",
    role=role,
    instance_type=processing_instance_type,
    instance_count=processing_instance_count,
    max_runtime_in_seconds=7200,
)
```

The model evaluation Processing Job will be running the Python code from the file src/evaluate_model_metrics.py. You can open and review the file.

Launch the Processing Job, passing the defined above parameters, custom script, path and the S3 bucket location of the test data.

In [30]:
```
from sagemaker.processing import ProcessingInput, ProcessingOutput

processor.run(
    code="src/evaluate_model_metrics.py",
    inputs=[
        ProcessingInput(
            input_name="model-tar-s3-uri",
            source=model_tar_s3_uri,
            destination="/opt/ml/processing/input/model/"
        ),
        ProcessingInput(
            input_name="evaluation-data-s3-uri",
            source=processed_test_data_s3_uri,
            destination="/opt/ml/processing/input/data/",
        ),
    ],
    outputs=[
        ProcessingOutput(s3_upload_mode="EndOfJob", output_name="metrics"
    ],
    arguments=["--max-seq-length", str(max_seq_length)],
    logs=True,
    wait=False,
)
```

```
Job Name:  sagemaker-scikit-learn-2021-09-16-09-27-40-683
Inputs:  [{'InputName': 'model-tar-s3-uri', 'AppManaged': False, 'S3Input
': {'S3Uri': 's3://sagemaker-us-east-1-593402094095/pytorch-training-2109
16-0857-002-fd13714c/output/model.tar.gz', 'LocalPath': '/opt/ml/processi
ng/input/model/', 'S3DataType': 'S3Prefix', 'S3InputMode': 'File', 'S3Dat
aDistributionType': 'FullyReplicated', 'S3CompressionType': 'None'}}, {'I
nputName': 'evaluation-data-s3-uri', 'AppManaged': False, 'S3Input': {'S3
Uri': 's3://sagemaker-us-east-1-593402094095/transformed/data/sentiment-t
est/', 'LocalPath': '/opt/ml/processing/input/data/', 'S3DataType': 'S3Pr
efix', 'S3InputMode': 'File', 'S3DataDistributionType': 'FullyReplicated'
, 'S3CompressionType': 'None'}}, {'InputName': 'code', 'AppManaged': Fals
e, 'S3Input': {'S3Uri': 's3://sagemaker-us-east-1-593402094095/sagemaker-
scikit-learn-2021-09-16-09-27-40-683/input/code/evaluate_model_metrics.py
', 'LocalPath': '/opt/ml/processing/input/code', 'S3DataType': 'S3Prefix'
, 'S3InputMode': 'File', 'S3DataDistributionType': 'FullyReplicated', 'S3
CompressionType': 'None'}}]
Outputs:  [{'OutputName': 'metrics', 'AppManaged': False, 'S3Output': {'S
3Uri': 's3://sagemaker-us-east-1-593402094095/sagemaker-scikit-learn-2021
-09-16-09-27-40-683/output/metrics', 'LocalPath': '/opt/ml/processing/out
put/metrics', 'S3UploadMode': 'EndOfJob'}}]
```

You can see the information about the Processing Jobs using the `describe` function. The result is in dictionary format. Let's pull the Processing Job name:

In [31]:
```python
scikit_processing_job_name = processor.jobs[-1].describe()["ProcessingJob

print('Processing Job name: {}'.format(scikit_processing_job_name))
```

```
Processing Job name: sagemaker-scikit-learn-2021-09-16-09-27-40-683
```

Exercise 7

Pull the Processing Job status from the Processing Job description.

Instructions: Print the keys of the Processing Job description dictionary, choose the one related to the status of the Processing Job and print the value of it.

In [32]:
```python
print(processor.jobs[-1].describe().keys())
```

```
dict_keys(['ProcessingInputs', 'ProcessingOutputConfig', 'ProcessingJobNa
me', 'ProcessingResources', 'StoppingCondition', 'AppSpecification', 'Rol
eArn', 'ProcessingJobArn', 'ProcessingJobStatus', 'LastModifiedTime', 'Cr
eationTime', 'ResponseMetadata'])
```

In [33]:
```python
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
scikit_processing_job_status = processor.jobs[-1].describe()['ProcessingJ
### END SOLUTION - DO NOT delete this comment for grading purposes
print('Processing job status: {}'.format(scikit_processing_job_status))
```

```
Processing job status: InProgress
```

Review the created Processing Job in the AWS console.

Instructions:

- open the link
- notice that you are in the section `Amazon SageMaker` -> `Processing Jobs`
- check the name of the Processing Job, its status and other available information

In [34]:
```
from IPython.core.display import display, HTML

display(
    HTML(
        '<b>Review <a target="blank" href="https://console.aws.amazon.com
            region, scikit_processing_job_name
        )
    )
)
```

Review Processing Job

Wait for about 5 minutes to review the CloudWatch Logs. You may open the file src/evaluate_model_metrics.py again and examine the outputs of the code in the CloudWatch Logs.

In [35]:
```
from IPython.core.display import display, HTML

display(
    HTML(
        '<b>Review <a target="blank" href="https://console.aws.amazon.com
            region, scikit_processing_job_name
        )
    )
)
```

Review CloudWatch Logs after about 5 minutes

After the completion of the Processing Job you can also review the output in the S3 bucket.

In [36]:
```
from IPython.core.display import display, HTML

display(
    HTML(
        '<b>Review <a target="blank" href="https://s3.console.aws.amazon.
            bucket, scikit_processing_job_name, region
        )
    )
)
```

Review S3 output data after the Processing Job has completed

Monitor the Processing Job:

In [37]:
```python
from pprint import pprint

running_processor = sagemaker.processing.ProcessingJob.from_processing_na
    processing_job_name=scikit_processing_job_name, sagemaker_session=ses
)

processing_job_description = running_processor.describe()

pprint(processing_job_description)
```

```
{'AppSpecification': {'ContainerArguments': ['--max-seq-length', '128'],
                      'ContainerEntrypoint': ['python3',
                                              '/opt/ml/processing/input/c
ode/evaluate_model_metrics.py'],
                      'ImageUri': '683313688378.dkr.ecr.us-east-1.amazona
ws.com/sagemaker-scikit-learn:0.23-1-cpu-py3'},
 'CreationTime': datetime.datetime(2021, 9, 16, 9, 27, 41, 193000, tzinfo
=tzlocal()),
 'LastModifiedTime': datetime.datetime(2021, 9, 16, 9, 27, 41, 610000, tz
info=tzlocal()),
 'ProcessingInputs': [{'AppManaged': False,
                       'InputName': 'model-tar-s3-uri',
                       'S3Input': {'LocalPath': '/opt/ml/processing/input
/model/',
                                   'S3CompressionType': 'None',
                                   'S3DataDistributionType': 'FullyReplic
ated',
                                   'S3DataType': 'S3Prefix',
                                   'S3InputMode': 'File',
                                   'S3Uri': 's3://sagemaker-us-east-1-593
402094095/pytorch-training-210916-0857-002-fd13714c/output/model.tar.gz'}
},
                      {'AppManaged': False,
                       'InputName': 'evaluation-data-s3-uri',
                       'S3Input': {'LocalPath': '/opt/ml/processing/input
/data/',
                                   'S3CompressionType': 'None',
                                   'S3DataDistributionType': 'FullyReplic
ated',
                                   'S3DataType': 'S3Prefix',
                                   'S3InputMode': 'File',
                                   'S3Uri': 's3://sagemaker-us-east-1-593
402094095/transformed/data/sentiment-test/'}},
                      {'AppManaged': False,
                       'InputName': 'code',
                       'S3Input': {'LocalPath': '/opt/ml/processing/input
/code',
                                   'S3CompressionType': 'None',
                                   'S3DataDistributionType': 'FullyReplic
ated',
                                   'S3DataType': 'S3Prefix',
                                   'S3InputMode': 'File',
                                   'S3Uri': 's3://sagemaker-us-east-1-593
402094095/sagemaker-scikit-learn-2021-09-16-09-27-40-683/input/code/evalu
ate_model_metrics.py'}}],
 'ProcessingJobArn': 'arn:aws:sagemaker:us-east-1:593402094095:processing
-job/sagemaker-scikit-learn-2021-09-16-09-27-40-683',
 'ProcessingJobName': 'sagemaker-scikit-learn-2021-09-16-09-27-40-683',
 'ProcessingJobStatus': 'InProgress',
 'ProcessingOutputConfig': {'Outputs': [{'AppManaged': False,
```

```
                                          'OutputName': 'metrics',
                                          'S3Output': {'LocalPath': '/opt/
ml/processing/output/metrics',
                                                       'S3UploadMode': 'En
dOfJob',
                                                       'S3Uri': 's3://sage
maker-us-east-1-593402094095/sagemaker-scikit-learn-2021-09-16-09-27-40-6
83/output/metrics'}}]},
 'ProcessingResources': {'ClusterConfig': {'InstanceCount': 1,
                                            'InstanceType': 'ml.c5.2xlarge
',
                                            'VolumeSizeInGB': 30}},
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '2289',
                                       'content-type': 'application/x-amz-
json-1.1',
                                       'date': 'Thu, 16 Sep 2021 09:27:41
GMT',
                                       'x-amzn-requestid': 'dab2cc22-65a8-
4118-b4e6-15e346939754'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'dab2cc22-65a8-4118-b4e6-15e346939754'
,
                      'RetryAttempts': 0},
 'RoleArn': 'arn:aws:iam::593402094095:role/c21581a444556l1011249t1w593-S
ageMakerExecutionRole-1A68B3IR8R29H',
 'StoppingCondition': {'MaxRuntimeInSeconds': 7200}}
```

Wait for the Processing Job to complete.

This cell will take approximately 5-10 minutes to run.

In [38]:
```
%%time

running_processor.wait(logs=False)
```

```
.........................................................................
...........................!CPU times: user 379 ms, sys: 62.3 ms, total:
441 ms
Wall time: 8min 22s
```

Please wait until ^^ Processing Job ^^ completes above

3.3. Inspect the processed output data

Let's take a look at the results of the Processing Job. Get the S3 bucket location of the output metrics:

In [39]:
```
processing_job_description = running_processor.describe()

output_config = processing_job_description["ProcessingOutputConfig"]
for output in output_config["Outputs"]:
    if output["OutputName"] == "metrics":
        processed_metrics_s3_uri = output["S3Output"]["S3Uri"]

print(processed_metrics_s3_uri)
```

```
s3://sagemaker-us-east-1-593402094095/sagemaker-scikit-learn-2021-09-16-09-27-40-683/output/metrics
```

List the content of the folder:

In [40]:
```
!aws s3 ls $processed_metrics_s3_uri/
```

```
2021-09-16 09:35:57      21158 confusion_matrix.png
2021-09-16 09:35:57         56 evaluation.json
```

The test accuracy can be pulled from the `evaluation.json` file.

In [41]:
```
import json
from pprint import pprint

metrics_json = sagemaker.s3.S3Downloader.read_file("{}/evaluation.json".f
    processed_metrics_s3_uri
))

print('Test accuracy: {}'.format(json.loads(metrics_json)))
```

```
Test accuracy: {'metrics': {'accuracy': {'value': 0.7249190938511327}}}
```

Copy image with the confusion matrix generated during the model evaluation into the folder `generated`.

In [42]:
```
!aws s3 cp $processed_metrics_s3_uri/confusion_matrix.png ./generated/

import time
time.sleep(10) # Slight delay for our notebook to recognize the newly-dow
```

```
download: s3://sagemaker-us-east-1-593402094095/sagemaker-scikit-learn-2021-09-16-09-27-40-683/output/metrics/confusion_matrix.png to generated/confusion_matrix.png
```

Show and review the confusion matrix, which is a table of all combinations of true (actual) and predicted labels. Each cell contains the number of the reviews for the corresponding sentiments. You can see that the highest numbers of the reviews appear in the diagonal cells, where the predicted sentiment equals the actual one.

In [43]:
```
%%html

<img src='./generated/confusion_matrix.png'>
```

Upload the notebook into S3 bucket for grading purposes.

Note: you may need to click on "Save" button before the upload.

In [44]:
```
!aws s3 cp ./C3_W1_Assignment.ipynb s3://$bucket/C3_W1_Assignment_Learner
```

```
upload: ./C3_W1_Assignment.ipynb to s3://sagemaker-us-east-1-593402094095/C3_W1_Assignment_Learner.ipynb
```

In [ ]:
```
```