Data labeling and human-in-the-loop pipelines with Amazon Augmented AI (A2I)

Introduction

In this lab you will create your own human workforce, a human task UI, and then define the human review workflow to perform data labeling. You will make the original predictions of the labels with the custom ML model, and then create a human loop if the probability scores are lower than the preset threshold. After the completion of the human loop tasks, you will review the results and prepare data for re-training.

Table of Contents

- 1. Set up Amazon Cognito user pool and define human workforce
  - 1.1. Create Amazon Cognito user pool
    - Exercise 1
  - 1.2. Create Amazon Cognito user pool client
    - Exercise 2
  - 1.3. Create Amazon Cognito user pool domain and group
    - Exercise 3
    - Exercise 4
  - 1.4. Create workforce and workteam
    - Exercise 5
    - Exercise 6
  - 1.5. Create an Amazon Cognito user and add the user to the group
- 2. Create Human Task UI
  - Exercise 7
- 3. Define human review workflow
  - Exercise 8
- 4. Start human loop with custom ML model
  - 4.1. Deploy a custom model
    - Exercise 9
  - 4.2. Start the human loop
    - Exercise 10
  - 4.3. Check status of the human loop
  - 4.4. Complete the human loop tasks
  - 4.5. Verify that the human loops were completed by the workforce
  - 4.6. View human labels and prepare the data for re-training

Let's install and import the required modules.

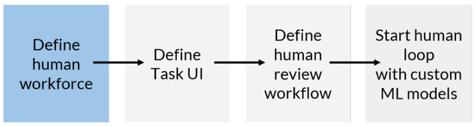
In [2]:
```
# please ignore warning messages during the installation
!pip install --disable-pip-version-check -q sagemaker==2.35.0
```

```
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 [3]:
```
import boto3
import sagemaker
import pandas as pd
from pprint import pprint
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c3/w3')

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

sm_runtime = boto3.client('sagemaker-runtime',
                          config=config)

sess = sagemaker.Session(sagemaker_client=sm,
                         sagemaker_runtime_client=sm_runtime)

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

s3 = boto3.Session().client(service_name='s3',
                            config=config)
cognito_idp = boto3.Session().client(service_name='cognito-idp',
                                     config=config)
a2i = boto3.Session().client(service_name='sagemaker-a2i-runtime',
                             config=config)
```

1. Set up Amazon Cognito user pool and define human workforce

The first step in the creation of the human-in-the-loop pipeline will be to create your own private workforce.


Define human workforce
Define Task UI
Define human review workflow
Start human loop with custom ML models

Amazon Cognito provides authentication, authorization, and user management for apps. This enables your workers to sign in directly to the labeling UI with a username and password.

You will construct an Amazon Cognito user pool, setting up its client, domain, and group. Then you'll create a SageMaker workforce, linking it to the Cognito user pool. Followed by the creation of a SageMaker workteam, linking it to the Cognito user pool and group. And finally, you will create a pool user and add it to the group.

To get started, let's construct the user pool and user pool client names.

In [4]:
```
import time
timestamp = int(time.time())

user_pool_name = 'groundtruth-user-pool-{}'.format(timestamp)
user_pool_client_name = 'groundtruth-user-pool-client-{}'.format(timestam

print("Amazon Cognito user pool name: {}".format(user_pool_name))
print("Amazon Cognito user pool client name: {}".format(user_pool_client_
```

```
Amazon Cognito user pool name: groundtruth-user-pool-1662439559
Amazon Cognito user pool client name: groundtruth-user-pool-client-166243
9559
```

1.1. Create Amazon Cognito user pool

Function `cognito_idp.create_user_pool` creates a new Amazon Cognito user pool. Passing the function result into a variable you can get the information about the response. The result is in dictionary format.

In [5]:
```
create_user_pool_response = cognito_idp.create_user_pool(PoolName=user_po
user_pool_id = create_user_pool_response['UserPool']['Id']

print("Amazon Cognito user pool ID: {}".format(user_pool_id))
```

```
Amazon Cognito user pool ID: us-east-1_pGwqg4SML
```

Exercise 1

Pull the Amazon Cognito user pool name from its description.

Instructions: Print the keys of the user pool, choose the one that corresponds to the name and print its value.

In [6]:
```
print(create_user_pool_response['UserPool'].keys())
```

```
dict_keys(['Id', 'Name', 'Policies', 'LambdaConfig', 'LastModifiedDate',
'CreationDate', 'SchemaAttributes', 'VerificationMessageTemplate', 'UserA
ttributeUpdateSettings', 'MfaConfiguration', 'EstimatedNumberOfUsers', 'E
mailConfiguration', 'AdminCreateUserConfig', 'Arn'])
```

In [8]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
user_pool_name = create_user_pool_response['UserPool']['Name'] # Replace
### END SOLUTION - DO NOT delete this comment for grading purposes
print('Amazon Cognito user pool name: {}'.format(user_pool_name))
```

```
Amazon Cognito user pool name: groundtruth-user-pool-1662439559
```

1.2. Create Amazon Cognito user pool client

Now let's set up the Amazon Cognito user pool client for the created above user pool.

The Amazon Cognito user pool client implements an open standard for authorization framework, `OAuth`. The standard enables apps to obtain limited access (scopes) to a user's data without giving away a user's password. It decouples authentication from authorization and supports multiple use cases addressing different device capabilities.

Exercise 2

Create the Amazon Cognito user pool client for the constructed user pool.

Instructions: Pass the user pool ID and the user pool client name into the function `cognito_idp.create_user_pool_client`. Review the other parameters of the function.

In [11]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
create_user_pool_client_response = cognito_idp.create_user_pool_client( #
    UserPoolId=user_pool_id, # Replace None
    ClientName=user_pool_client_name, # Replace None
### END SOLUTION - DO NOT delete this comment for grading purposes
    GenerateSecret=True, # boolean to specify whether you want to generat
    # a list of provider names for the identity providers that are suppor
    SupportedIdentityProviders=[
        'COGNITO'
    ],
    # a list of the allowed OAuth flows, e.g. code, implicit, client_cred
    AllowedOAuthFlows=[
        'code',
        'implicit'
    ],
    # a list of the allowed OAuth scopes, e.g. phone, email, openid, and
    AllowedOAuthScopes=[
        'email',
        'openid',
        'profile'
    ],
    # a list of allowed redirect (callback) URLs for the identity provide
    CallbackURLs=[
        'https://datascienceonaws.com',
    ],
    # set to true if the client is allowed to follow the OAuth protocol w
    AllowedOAuthFlowsUserPoolClient=True
)

client_id = create_user_pool_client_response['UserPoolClient']['ClientId'
print('Amazon Cognito user pool client ID: {}'.format(client_id))
```

```
Amazon Cognito user pool client ID: 30h413cikj3ths46dampises92
```

1.3. Create Amazon Cognito user pool domain and group

Exercise 3

Set up the Amazon Cognito user pool domain for the constructed user pool.

Instructions: Pass the user pool ID and the user pool domain name into the function `cognito_idp.create_user_pool_domain`.

In [12]:
```
user_pool_domain_name = 'groundtruth-user-pool-domain-{}'.format(timestam

try:
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    cognito_idp.create_user_pool_domain( # Replace None
        UserPoolId=user_pool_id, # Replace None
        Domain=user_pool_domain_name # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    )
    print("Created Amazon Cognito user pool domain: {}".format(user_pool_
except:
    print("Amazon Cognito user pool domain {} already exists".format(user
Created Amazon Cognito user pool domain: groundtruth-user-pool-domain-166
2439559
```

You will use the following function to check if the Amazon Cognito user group already exists.

In [13]:
```python
def check_user_pool_group_existence(user_pool_id, user_pool_group_name):
    for group in cognito_idp.list_groups(UserPoolId=user_pool_id)['Groups
        if user_pool_group_name == group['GroupName']:
            return True
    return False
```

Exercise 4

Set up Amazon Cognito user group.

Instructions: Pass the user pool ID and the user group name into the function `cognito_idp.create_group`.

In [14]:
```python
user_pool_group_name = 'groundtruth-user-pool-group-{}'.format(timestamp)

if not check_user_pool_group_existence(user_pool_id, user_pool_group_name
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    cognito_idp.create_group( # Replace None
        UserPoolId=user_pool_id, # Replace None
        GroupName=user_pool_group_name # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    )
    print("Created Amazon Cognito user group: {}".format(user_pool_group_
else:
    print("Amazon Cognito user group {} already exists".format(user_pool_
```

```
Created Amazon Cognito user group: groundtruth-user-pool-group-1662439559
```

1.4. Create workforce and workteam

Use the following function to check if the workforce already exists. You can only create one workforce per region, therefore you'll have to delete any other existing workforce, together with all of the related workteams.

In [15]:
```python
def check_workforce_existence(workforce_name):
    for workforce in sm.list_workforces()['Workforces']:
        if workforce_name == workforce['WorkforceName']:
            return True
        else:
            for workteam in sm.list_workteams()['Workteams']:
                sm.delete_workteam(WorkteamName=workteam['WorkteamName'])
            sm.delete_workforce(WorkforceName=workforce['WorkforceName'])
    return False
```

Exercise 5

Create a workforce.

Instructions: Pass the Amazon Cognito user pool ID and client ID into the Cognito configuration of the function `sm.create_workforce`.

In [16]:
```
workforce_name = 'groundtruth-workforce-name-{}'.format(timestamp)

if not check_workforce_existence(workforce_name):
    create_workforce_response = sm.create_workforce(
        WorkforceName=workforce_name,
        CognitoConfig={
            ### BEGIN SOLUTION - DO NOT delete this comment for grading p
            'UserPool': user_pool_id, # Replace None
            'ClientId': client_id # Replace None
            ### END SOLUTION - DO NOT delete this comment for grading pur
        }
    )
    print("Workforce name: {}".format(workforce_name))
    pprint(create_workforce_response)
else:
    print("Workforce {} already exists".format(workforce_name))
```

```
Workforce name: groundtruth-workforce-name-1662439559
{'ResponseMetadata': {'HTTPHeaders': {'content-length': '107',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Tue, 06 Sep 2022 04:54:35
GMT',
                                      'x-amzn-requestid': '407106a9-71e1-
4d41-928c-1aac012bd714'},
                      'HTTPStatusCode': 200,
                      'RequestId': '407106a9-71e1-4d41-928c-1aac012bd714'
,
                      'RetryAttempts': 0},
 'WorkforceArn': 'arn:aws:sagemaker:us-east-1:395611198364:workforce/grou
ndtruth-workforce-name-1662439559'}
```

You can use the `sm.describe_workforce` function to get the information about the workforce.

In [17]:
```
describe_workforce_response = sm.describe_workforce(WorkforceName=workfor
describe_workforce_response
```

```
Out[17]: {'Workforce': {'WorkforceName': 'groundtruth-workforce-name-1662439559',
           'WorkforceArn': 'arn:aws:sagemaker:us-east-1:395611198364:workforce/gro
         undtruth-workforce-name-1662439559',
           'LastUpdatedDate': datetime.datetime(2022, 9, 6, 4, 54, 35, 555000, tzi
         nfo=tzlocal()),
           'SourceIpConfig': {'Cidrs': []},
           'SubDomain': '4gb4gzwziq.labeling.us-east-1.sagemaker.aws',
           'CognitoConfig': {'UserPool': 'us-east-1_pGwqg4SML',
            'ClientId': '30h413cikj3ths46dampises92'},
           'CreateDate': datetime.datetime(2022, 9, 6, 4, 54, 34, 931000, tzinfo=t
         zlocal()),
           'Status': 'Initializing'},
          'ResponseMetadata': {'RequestId': 'ba3a121f-d203-4b13-b723-3d7fc74201b8'
         ,
           'HTTPStatusCode': 200,
           'HTTPHeaders': {'x-amzn-requestid': 'ba3a121f-d203-4b13-b723-3d7fc74201
         b8',
            'content-type': 'application/x-amz-json-1.1',
            'content-length': '445',
            'date': 'Tue, 06 Sep 2022 04:54:38 GMT'},
           'RetryAttempts': 0}}
```

Use the following function to check if the workteam already exists. If there are no workteams in the list, give some time for the workforce to set up.

```
In [18]: def check_workteam_existence(workteam_name):
             if sm.list_workteams()['Workteams']:
                 for workteam in sm.list_workteams()['Workteams']:
                     if workteam_name == workteam['WorkteamName']:
                         return True
             else:
                 time.sleep(60)
                 return False
             return False
```

Exercise 6

Create a workteam.

Instructions: Pass the Amazon Cognito user pool ID, client ID, and group name into the Cognito member definition of the function `sm.create_workteam`.

This cell may take 1-2 minutes to run.

```
In [19]: workteam_name = 'groundtruth-workteam-{}'.format(timestamp)

         if not check_workteam_existence(workteam_name):
             create_workteam_response = sm.create_workteam(
                 Description='groundtruth workteam',
                 WorkforceName=workforce_name,
                 WorkteamName=workteam_name,
                 # objects that identify the workers that make up the work team
                 MemberDefinitions=[{
                     'CognitoMemberDefinition': {
                         ### BEGIN SOLUTION - DO NOT delete this comment for gradi
                         'UserPool': user_pool_id, # Replace None
                         'ClientId': client_id, # Replace None
                         'UserGroup': user_pool_group_name # Replace None
                         ### END SOLUTION - DO NOT delete this comment for grading
                     }
                 }]
             )
             pprint(create_workteam_response)
         else:
             print("Workteam {} already exists".format(workteam_name))
```

```
{'ResponseMetadata': {'HTTPHeaders': {'content-length': '113',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Tue, 06 Sep 2022 04:56:28
GMT',
                                      'x-amzn-requestid': '0448e198-b9f5-
44d1-8653-27b3d3db2295'},
                      'HTTPStatusCode': 200,
                      'RequestId': '0448e198-b9f5-44d1-8653-27b3d3db2295'
,
                      'RetryAttempts': 0},
 'WorkteamArn': 'arn:aws:sagemaker:us-east-1:395611198364:workteam/privat
e-crowd/groundtruth-workteam-1662439559'}
```

You can use `sm.describe_workteam` function to get information about the workteam.

```
In [20]: describe_workteam_response = sm.describe_workteam(WorkteamName=workteam_n
         describe_workteam_response
```

Out[20]:

```
{'Workteam': {'WorkteamName': 'groundtruth-workteam-1662439559',
  'MemberDefinitions': [{'CognitoMemberDefinition': {'UserPool': 'us-east
-1_pGwqg4SML',
     'UserGroup': 'groundtruth-user-pool-group-1662439559',
     'ClientId': '30h413cikj3ths46dampises92'}}],
  'WorkteamArn': 'arn:aws:sagemaker:us-east-1:395611198364:workteam/priva
te-crowd/groundtruth-workteam-1662439559',
  'Description': 'groundtruth workteam',
  'SubDomain': '4gb4gzwziq.labeling.us-east-1.sagemaker.aws',
  'CreateDate': datetime.datetime(2022, 9, 6, 4, 56, 28, 118000, tzinfo=t
zlocal()),
  'LastUpdatedDate': datetime.datetime(2022, 9, 6, 4, 56, 28, 937000, tzi
nfo=tzlocal()),
  'NotificationConfiguration': {}},
 'ResponseMetadata': {'RequestId': '5c25dc61-5ea4-4f70-b0eb-783d52efcf8f'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '5c25dc61-5ea4-4f70-b0eb-783d52efcf
8f',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '544',
   'date': 'Tue, 06 Sep 2022 04:56:54 GMT'},
  'RetryAttempts': 0}}
```

Now you can pull the workteam ARN either from `create_workteam_response` or `describe_workteam_response`.

In [21]:

```
workteam_arn = describe_workteam_response['Workteam']['WorkteamArn']
workteam_arn
```

Out[21]:

```
'arn:aws:sagemaker:us-east-1:395611198364:workteam/private-crowd/groundtr
uth-workteam-1662439559'
```

Review the created workteam in the AWS console.

Instructions:

- open the link
- notice that you are in the section Amazon SageMaker -> Labeling workforces
- check the name of the workteam, its Amazon Cognito user group and ARN

In [22]:

```
from IPython.core.display import display, HTML

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

Review workteam

1.5. Create an Amazon Cognito user and add the user to the group

Use the following function to check if the Amazon Cognito user already exists.

In [23]:
```
def check_user_existence(user_pool_id, user_name):
    for user in cognito_idp.list_users(UserPoolId=user_pool_id)['Users']:
        if user_name == user['Username']:
            return True
    return False
```

Create a user passing the username, temporary password, and the Amazon Cognito user pool ID.

In [24]:
```
user_name = 'user-{}'.format(timestamp)

temporary_password = 'Password@420'

if not check_user_existence(user_pool_id, user_name):
    create_user_response=cognito_idp.admin_create_user(
        Username=user_name,
        UserPoolId=user_pool_id,
        TemporaryPassword=temporary_password,
        MessageAction='SUPPRESS' # suppress sending the invitation messag
    )
    pprint(create_user_response)
else:
    print("Amazon Cognito user {} already exists".format(user_name))
```

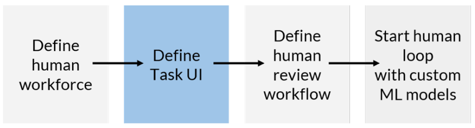
```
{'ResponseMetadata': {'HTTPHeaders': {'connection': 'keep-alive',
                                      'content-length': '242',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Tue, 06 Sep 2022 04:57:16
GMT',
                                      'x-amzn-requestid': '6c1fd9f5-6d01-
441d-86ed-c62ad0665c6a'},
                      'HTTPStatusCode': 200,
                      'RequestId': '6c1fd9f5-6d01-441d-86ed-c62ad0665c6a'
,
                      'RetryAttempts': 0},
 'User': {'Attributes': [{'Name': 'sub',
                          'Value': '04fa4b1d-738d-449a-838f-340797ca6324'
}],
          'Enabled': True,
          'UserCreateDate': datetime.datetime(2022, 9, 6, 4, 57, 16, 2330
00, tzinfo=tzlocal()),
          'UserLastModifiedDate': datetime.datetime(2022, 9, 6, 4, 57, 16
, 233000, tzinfo=tzlocal()),
          'UserStatus': 'FORCE_CHANGE_PASSWORD',
          'Username': 'user-1662439559'}}
```

Add the user into the Amazon Cognito user group.

In [25]:
```
cognito_idp.admin_add_user_to_group(
    UserPoolId=user_pool_id,
    Username=user_name,
    GroupName=user_pool_group_name
)
```

Out[25]:
```
{'ResponseMetadata': {'RequestId': 'cbc66e5e-81bb-4aeb-8736-6f20a2ac4ccd'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'date': 'Tue, 06 Sep 2022 04:57:19 GMT',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '0',
   'connection': 'keep-alive',
   'x-amzn-requestid': 'cbc66e5e-81bb-4aeb-8736-6f20a2ac4ccd'},
  'RetryAttempts': 0}}
```

2. Create Human Task UI


Define human workforce
Define Task UI
Define human review workflow
Start human loop with custom ML models

Create a Human Task UI resource, using a worker task UI template. This template will be rendered to the human workers whenever human interaction is required.

Below there is a simple demo template provided, that is compatible with the current use case of classifying product reviews into the three sentiment classes. For other pre-built UIs (there are 70+), check: https://github.com/aws-samples/amazon-a2i-sample-task-uis

In [27]:
```
template = r"""
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
    <crowd-classifier name="sentiment"
                      categories="['-1', '0', '1']"
                      initial-value="{{ task.input.initialValue }}"
                      header="Classify Reviews into Sentiment:  -1 (negat

        <classification-target>
            {{ task.input.taskObject }}
        </classification-target>

        <full-instructions header="Classify reviews into sentiment:  -1 (
            <p><strong>1</strong>: joy, excitement, delight</p>
            <p><strong>0</strong>: neither positive or negative, such as
            <p><strong>-1</strong>: anger, sarcasm, anxiety</p>
        </full-instructions>

        <short-instructions>
            Classify reviews into sentiment:  -1 (negative), 0 (neutral),
        </short-instructions>
    </crowd-classifier>
</crowd-form>
"""
```

Exercise 7

Create a human task UI resource.

Instructions: Pass the worker task UI template defined above as the content of the UI template parameter to the function `sm.create_human_task_ui`.

In [28]:
```
# Task UI name - this value is unique per account and region. You can als
task_ui_name = 'ui-{}'.format(timestamp)

human_task_ui_response = sm.create_human_task_ui(
    HumanTaskUiName=task_ui_name,
    UiTemplate={
        ### BEGIN SOLUTION - DO NOT delete this comment for grading purpo
        "Content": template  # Replace None
        ### END SOLUTION - DO NOT delete this comment for grading purpose
    }
)
human_task_ui_response
```

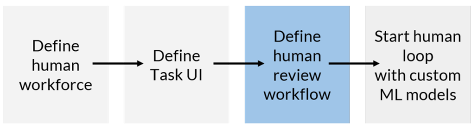
Out[28]:
```
{'HumanTaskUiArn': 'arn:aws:sagemaker:us-east-1:395611198364:human-task-u
i/ui-1662439559',
 'ResponseMetadata': {'RequestId': 'd4fe3452-949a-4e09-86ae-78afad5b34f8'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'd4fe3452-949a-4e09-86ae-78afad5b34
f8',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '89',
   'date': 'Tue, 06 Sep 2022 05:00:35 GMT'},
  'RetryAttempts': 0}}
```

Pull the ARN of the human task UI:

In [29]:
```
human_task_ui_arn = human_task_ui_response["HumanTaskUiArn"]
print(human_task_ui_arn)
```

```
arn:aws:sagemaker:us-east-1:395611198364:human-task-ui/ui-1662439559
```

3. Define human review workflow


Define human workforce
Define Task UI
Define human review workflow
Start human loop with custom ML models

In this section, you are going to create a Flow Definition. Flow Definitions allow you to specify:

- The workforce (in fact, it is a workteam) that your tasks will be sent to.
- The instructions that your workforce will receive (worker task template).
- The configuration of your worker tasks, including the number of workers that receive a task and time limits to complete tasks.
- Where your output data will be stored.

Here you are going to use the API, but you can optionally create this workflow definition in the console as well.

For more details and instructions, see: https://docs.aws.amazon.com/sagemaker/latest/dg/a2i-create-flow-definition.html.

Let's construct the S3 bucket output path.

In [30]:
```
output_path = 's3://{}/a2i-results-{}'.format(bucket, timestamp)
print(output_path)
```

```
s3://sagemaker-us-east-1-395611198364/a2i-results-1662439559
```

Exercise 8

Construct the Flow Definition with the workteam and human task UI in the human loop configurations that you created above.

Instructions: Pass the workteam and human task UI ARNs into the `HumanLoopConfig` dictionary within the function `sm.create_flow_definition`. Review the other parameters.

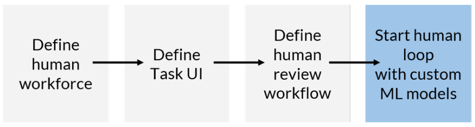
In [31]:
```
# Flow definition name - this value is unique per account and region
flow_definition_name = 'fd-{}'.format(timestamp)

create_workflow_definition_response = sm.create_flow_definition(
    FlowDefinitionName=flow_definition_name,
    RoleArn=role,
    HumanLoopConfig={
        ### BEGIN SOLUTION - DO NOT delete this comment for grading purpo
        "WorkteamArn": workteam_arn, # Replace None
        "HumanTaskUiArn": human_task_ui_arn, # Replace None
        ### END SOLUTION - DO NOT delete this comment for grading purpose
        "TaskCount": 1, # the number of workers that receive a task
        "TaskDescription": "Classify Reviews into sentiment:  -1 (negativ
        "TaskTitle": "Classify Reviews into sentiment:  -1 (negative), 0
    },
    OutputConfig={"S3OutputPath": output_path},
)

augmented_ai_flow_definition_arn = create_workflow_definition_response["F
```

You can pull information about the Flow Definition with the function `sm.describe_flow_definition` and wait for its status value `FlowDefinitionStatus` to become `Active`.

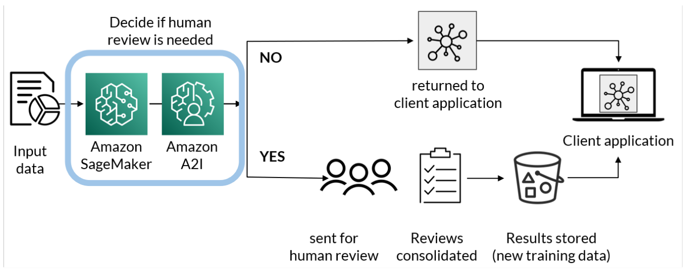
In [32]:
```python
for _ in range(60):
    describe_flow_definition_response = sm.describe_flow_definition(FlowD
    print(describe_flow_definition_response["FlowDefinitionStatus"])
    if describe_flow_definition_response["FlowDefinitionStatus"] == "Acti
        print("Flow Definition is active")
        break
    time.sleep(2)
```

```
Active
Flow Definition is active
```

4. Start human loop with custom ML model


Define human workforce
Define Task UI
Define human review workflow
Start human loop with custom ML models

Deploy a custom ML model into an endpoint and call it to predict labels for some sample reviews. Check the confidence score for each prediction. If it is smaller than the threshold, engage your workforce for a human review, starting a human loop. Fix the labels by completing the human loop tasks and review the results.



4.1. Deploy a custom model

Set up a sentiment predictor class to be wrapped later into the PyTorch Model.

Exercise 9

Create a Sentiment Predictor class.

Instructions: Pass the JSON serializer and deserializer objects here, calling them with the functions `JSONLinesSerializer()` and `JSONLinesDeserializer()`, respectively. More information about the serializers can be found here.

```
In [33]: from sagemaker.predictor import Predictor
         from sagemaker.serializers import JSONLinesSerializer
         from sagemaker.deserializers import JSONLinesDeserializer

         class SentimentPredictor(Predictor):
             def __init__(self, endpoint_name, sagemaker_session):
                 super().__init__(
                     endpoint_name,
                     sagemaker_session=sagemaker_session,
                     ### BEGIN SOLUTION - DO NOT delete this comment for grading p
                     serializer=JSONLinesSerializer(), # Replace None
                     deserializer=JSONLinesDeserializer() # Replace None
                     ### END SOLUTION - DO NOT delete this comment for grading pur
                 )
```

Create a SageMaker model based on the model artifact saved in the S3 bucket.

```
In [34]: from sagemaker.pytorch.model import PyTorchModel

         pytorch_model_name = 'model-{}'.format(timestamp)

         model = PyTorchModel(name=pytorch_model_name,
                              model_data='s3://dlai-practical-data-science/models/
                              predictor_cls=SentimentPredictor,
                              entry_point='inference.py',
                              source_dir='src',
                              framework_version='1.6.0',
                              py_version='py3',
                              role=role)
```

Now you will create a SageMaker Endpoint from the model. 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.

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

```
In [35]: %%time

         pytorch_endpoint_name = 'endpoint-{}'.format(timestamp)

         predictor = model.deploy(initial_instance_count=1,
                                  instance_type='ml.m5.large',
                                  endpoint_name=pytorch_endpoint_name)
```

```
----------!CPU times: user 2min 15s, sys: 9.85 s, total: 2min 24s
Wall time: 7min 22s
```

You can review the endpoint in the AWS console and check its status.

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

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

Review SageMaker REST Endpoint

4.2. Start the human loop

Let's create a list of sample reviews.

```
In [37]: reviews = ["I enjoy this product",
                     "I am unhappy with this product",
                     "It is okay",
                     "sometimes it works"]
```

Now you can send each of the sample reviews to the model via the `predictor.predict()` API call. Note that you need to pass the reviews in the JSON format that model expects as input. Then, you parse the model's response to obtain the predicted label and the confidence score.

After that, you check the condition for when you want to engage a human for review. You can check whether the returned confidence score is under the defined threshold of 90%, which would mean that you would want to start the human loop with the predicted label and the review as inputs. Finally, you start the human loop passing the input content and Flow Definition defined above.

Exercise 10

Complete the dictionary `input_content`, which should contain the original prediction (`'initialValue'` key) and review text (`'taskObject'` key).

In [38]:
```
import json

human_loops_started = []

CONFIDENCE_SCORE_THRESHOLD = 0.90

for review in reviews:
    inputs = [
        {"features": [review]},
    ]

    response = predictor.predict(inputs)
    print(response)
    prediction = response[0]['predicted_label']
    confidence_score = response[0]['probability']

    print('Checking prediction confidence {} for sample review: "{}"'.for

    # condition for when you want to engage a human for review
    if confidence_score < CONFIDENCE_SCORE_THRESHOLD:
        human_loop_name = str(time.time()).replace('.', '-') # using mill
        input_content = {
            ### BEGIN SOLUTION - DO NOT delete this comment for grading p
            "initialValue": prediction, # Replace None
            "taskObject": review # Replace None
            ### END SOLUTION - DO NOT delete this comment for grading pur
        }
        start_loop_response = a2i.start_human_loop(
            HumanLoopName=human_loop_name,
            FlowDefinitionArn=augmented_ai_flow_definition_arn,
            HumanLoopInput={"InputContent": json.dumps(input_content)},
        )

        human_loops_started.append(human_loop_name)

        print(
            f"Confidence score of {confidence_score * 100}% for predictio
        )
        print(f"*** ==> Starting human loop with name: {human_loop_name}
    else:
        print(
            f"Confidence score of {confidence_score * 100}% for star rati
        )
        print("Human loop not needed. \n")
```

```
[{'probability': 0.9376369118690491, 'predicted_label': 1}]
Checking prediction confidence 0.9376369118690491 for sample review: "I e
njoy this product"
Confidence score of 93.76369118690491% for star rating of 1 is above thre
shold of 90.0%
Human loop not needed.

[{'probability': 0.6340296864509583, 'predicted_label': -1}]
Checking prediction confidence 0.6340296864509583 for sample review: "I a
m unhappy with this product"
Confidence score of 63.402968645095825% for prediction of -1 is less than
the threshold of 90.0%
*** ==> Starting human loop with name: 1662441389-4061365

[{'probability': 0.5422114729881287, 'predicted_label': 1}]
Checking prediction confidence 0.5422114729881287 for sample review: "It
is okay"
Confidence score of 54.221147298812866% for prediction of 1 is less than
the threshold of 90.0%
*** ==> Starting human loop with name: 1662441389-9554596

[{'probability': 0.3931102454662323, 'predicted_label': 1}]
Checking prediction confidence 0.3931102454662323 for sample review: "som
etimes it works"
Confidence score of 39.31102454662323% for prediction of 1 is less than t
he threshold of 90.0%
*** ==> Starting human loop with name: 1662441390-4259758
```

Review the results above. Three of the sample reviews with the probability scores lower than the threshold went into the human loop. The original predicted labels are passed together with the review text and will be seen in the task.

4.3. Check status of the human loop

Function `a2i.describe_human_loop` can be used to pull the information about the human loop.

In [39]:
```python
completed_human_loops = []
for human_loop_name in human_loops_started:
    resp = a2i.describe_human_loop(HumanLoopName=human_loop_name)
    print(f"HumanLoop Name: {human_loop_name}")
    print(f'HumanLoop Status: {resp["HumanLoopStatus"]}')
    print(f'HumanLoop Output Destination: {resp["HumanLoopOutput"]}')
    print("")

    if resp["HumanLoopStatus"] == "Completed":
        completed_human_loops.append(resp)
```

```
HumanLoop Name: 1662441389-4061365
HumanLoop Status: InProgress
HumanLoop Output Destination: {'OutputS3Uri': 's3://sagemaker-us-east-1-3
95611198364/a2i-results-1662439559/fd-1662439559/2022/09/06/05/16/29/1662
441389-4061365/output.json'}

HumanLoop Name: 1662441389-9554596
HumanLoop Status: InProgress
HumanLoop Output Destination: {'OutputS3Uri': 's3://sagemaker-us-east-1-3
95611198364/a2i-results-1662439559/fd-1662439559/2022/09/06/05/16/30/1662
441389-9554596/output.json'}

HumanLoop Name: 1662441390-4259758
HumanLoop Status: InProgress
HumanLoop Output Destination: {'OutputS3Uri': 's3://sagemaker-us-east-1-3
95611198364/a2i-results-1662439559/fd-1662439559/2022/09/06/05/16/30/1662
441390-4259758/output.json'}
```

4.4. Complete the human loop tasks

Pull labeling UI from the workteam information to get into the human loop tasks in the AWS console.

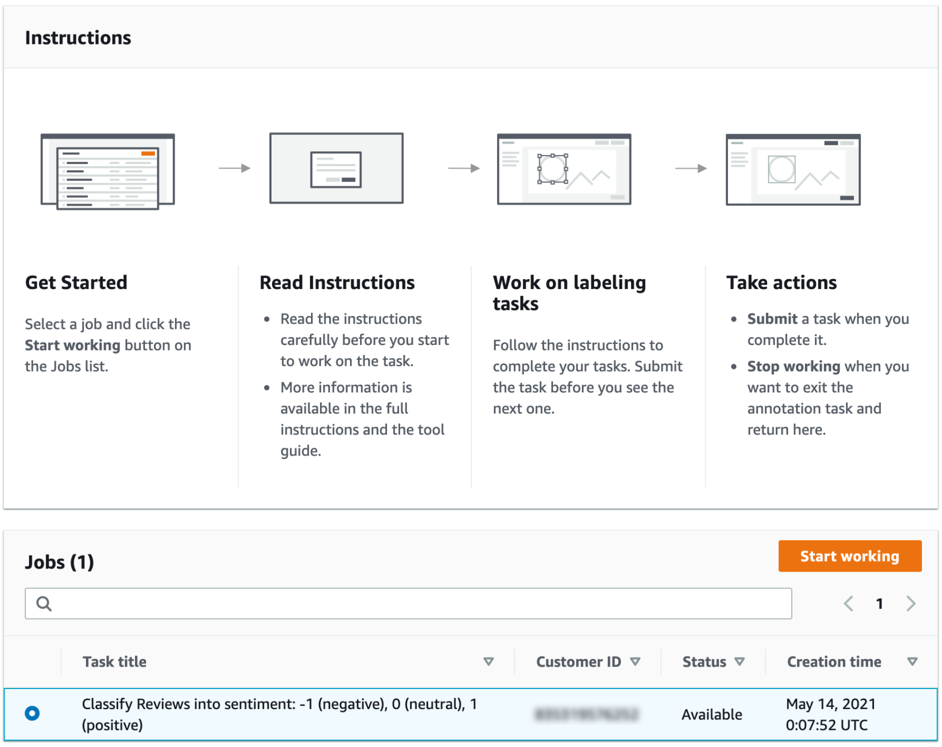
In [40]:
```
labeling_ui = sm.describe_workteam(WorkteamName=workteam_name)["Workteam"
print(labeling_ui)
```

```
4gb4gzwziq.labeling.us-east-1.sagemaker.aws
```

Navigate to the link below and login with the defined username and password. Complete the human loop following the provided instructions.

In [41]:
```
from IPython.core.display import display, HTML

display(HTML('Click <a target="blank" href="https://{}"><b>here</b></a> t
```

Click here to start labeling with username **user-1662439559** and temporary password **Password@420**



Get Started

Select a job and click the **Start working** button on the Jobs list.

Read Instructions

- Read the instructions carefully before you start to work on the task.
- More information is available in the full instructions and the tool guide.

Work on labeling tasks

Follow the instructions to complete your tasks. Submit the task before you see the next one.

Take actions

- **Submit** a task when you complete it.
- **Stop working** when you want to exit the annotation task and return here.

Jobs (1) Start working

	Task title	Customer ID	Status	Creation time
●	Classify Reviews into sentiment: -1 (negative), 0 (neutral), 1 (positive)	[redacted]	Available	May 14, 2021 0:07:52 UTC

*Wait for workers to complete ^^ their human loop tasks ^^*

4.5. Verify that the human loops were completed by the workforce

Note: This cell will not complete until you label the data following the instructions above.

In [42]:
```python
import time

completed_human_loops = []
for human_loop_name in human_loops_started:
    resp = a2i.describe_human_loop(HumanLoopName=human_loop_name)
    print(f"HumanLoop Name: {human_loop_name}")
    print(f'HumanLoop Status: {resp["HumanLoopStatus"]}')
    print(f'HumanLoop Output Destination: {resp["HumanLoopOutput"]}')
    print("")
    while resp["HumanLoopStatus"] != "Completed":
        print(f"Waiting for HumanLoop to complete.")
        time.sleep(10)
        resp = a2i.describe_human_loop(HumanLoopName=human_loop_name)
    if resp["HumanLoopStatus"] == "Completed":
        completed_human_loops.append(resp)
        print(f"Completed!")
        print("")
```

```
HumanLoop Name: 1662441389-4061365
HumanLoop Status: Completed
HumanLoop Output Destination: {'OutputS3Uri': 's3://sagemaker-us-east-1-3
95611198364/a2i-results-1662439559/fd-1662439559/2022/09/06/05/16/29/1662
441389-4061365/output.json'}

Completed!

HumanLoop Name: 1662441389-9554596
HumanLoop Status: Completed
HumanLoop Output Destination: {'OutputS3Uri': 's3://sagemaker-us-east-1-3
95611198364/a2i-results-1662439559/fd-1662439559/2022/09/06/05/16/30/1662
441389-9554596/output.json'}

Completed!

HumanLoop Name: 1662441390-4259758
HumanLoop Status: Completed
HumanLoop Output Destination: {'OutputS3Uri': 's3://sagemaker-us-east-1-3
95611198364/a2i-results-1662439559/fd-1662439559/2022/09/06/05/16/30/1662
441390-4259758/output.json'}

Completed!
```

Note: This cell ^^ above ^^ will not complete until you label the data following the instructions above.

4.6. View human labels and prepare the data for re-training

Once the work is complete, Amazon A2I stores the results in the specified S3 bucket and sends a Cloudwatch Event. Let's check the S3 contents.

In [43]:
```
import re
from pprint import pprint

fixed_items = []

for resp in completed_human_loops:
    split_string = re.split("s3://" + bucket + "/", resp["HumanLoopOutput
    output_bucket_key = split_string[1]

    response = s3.get_object(Bucket=bucket, Key=output_bucket_key)
    content = response["Body"].read().decode("utf-8")
    json_output = json.loads(content)
    pprint(json_output)

    input_content = json_output["inputContent"]
    human_answer = json_output["humanAnswers"][0]["answerContent"]
    fixed_item = {"input_content": input_content, "human_answer": human_a
    fixed_items.append(fixed_item)
```

```
{'flowDefinitionArn': 'arn:aws:sagemaker:us-east-1:395611198364:flow-defi
nition/fd-1662439559',
 'humanAnswers': [{'acceptanceTime': '2022-09-06T05:18:56.937Z',
                   'answerContent': {'sentiment': {'label': '-1'}},
                   'submissionTime': '2022-09-06T05:19:02.373Z',
                   'timeSpentInSeconds': 5.436,
                   'workerId': '2be14b5b856a2aca',
                   'workerMetadata': {'identityData': {'identityProviderT
ype': 'Cognito',
                                                       'issuer': 'https:/
/cognito-idp.us-east-1.amazonaws.com/us-east-1_pGwqg4SML',
                                                       'sub': '04fa4b1d-7
38d-449a-838f-340797ca6324'}}}],
 'humanLoopName': '1662441389-4061365',
 'inputContent': {'initialValue': -1,
                  'taskObject': 'I am unhappy with this product'}}
{'flowDefinitionArn': 'arn:aws:sagemaker:us-east-1:395611198364:flow-defi
nition/fd-1662439559',
 'humanAnswers': [{'acceptanceTime': '2022-09-06T05:18:25.076Z',
                   'answerContent': {'sentiment': {'label': '0'}},
                   'submissionTime': '2022-09-06T05:18:49.678Z',
                   'timeSpentInSeconds': 24.602,
                   'workerId': '2be14b5b856a2aca',
                   'workerMetadata': {'identityData': {'identityProviderT
ype': 'Cognito',
                                                       'issuer': 'https:/
/cognito-idp.us-east-1.amazonaws.com/us-east-1_pGwqg4SML',
                                                       'sub': '04fa4b1d-7
38d-449a-838f-340797ca6324'}}}],
 'humanLoopName': '1662441389-9554596',
 'inputContent': {'initialValue': 1, 'taskObject': 'It is okay'}}
{'flowDefinitionArn': 'arn:aws:sagemaker:us-east-1:395611198364:flow-defi
nition/fd-1662439559',
 'humanAnswers': [{'acceptanceTime': '2022-09-06T05:18:49.749Z',
                   'answerContent': {'sentiment': {'label': '0'}},
                   'submissionTime': '2022-09-06T05:18:56.869Z',
                   'timeSpentInSeconds': 7.12,
                   'workerId': '2be14b5b856a2aca',
                   'workerMetadata': {'identityData': {'identityProviderT
ype': 'Cognito',
                                                       'issuer': 'https:/
/cognito-idp.us-east-1.amazonaws.com/us-east-1_pGwqg4SML',
                                                       'sub': '04fa4b1d-7
38d-449a-838f-340797ca6324'}}}],
 'humanLoopName': '1662441390-4259758',
 'inputContent': {'initialValue': 1, 'taskObject': 'sometimes it works'}}
```

Now you can prepare the data for re-training.

In [44]:
```python
df_fixed_items = pd.DataFrame(fixed_items)
df_fixed_items.head()
```

Out[44]:

	input_content	human_answer
0	{'initialValue': -1, 'taskObject': 'I am unhap...	{'sentiment': {'label': '-1'}}
1	{'initialValue': 1, 'taskObject': 'It is okay'}	{'sentiment': {'label': '0'}}
2	{'initialValue': 1, 'taskObject': 'sometimes i...	{'sentiment': {'label': '0'}}

Upload the notebook into S3 bucket for grading purposes.

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

In [45]:
```
!aws s3 cp ./C3_W3_Assignment.ipynb s3://$bucket/C3_W3_Assignment_Learner
```

```
upload: ./C3_W3_Assignment.ipynb to s3://sagemaker-us-east-1-395611198364
/C3_W3_Assignment_Learner.ipynb
```

In [ ]:
```

```