PRACTICAL DATA SCIENCE ON AWS

DeepLearning.AI

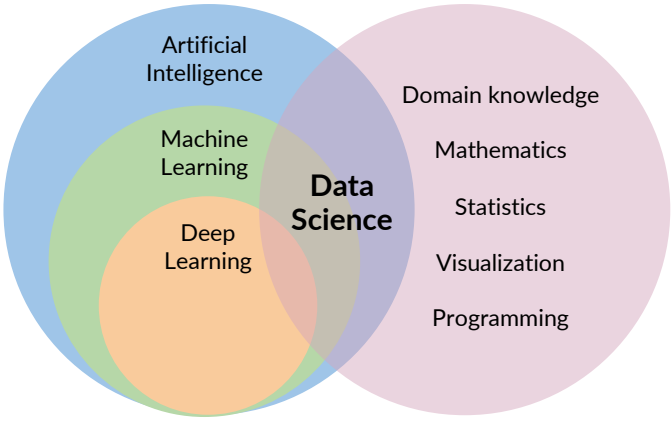
DeepLearning.AI

aws

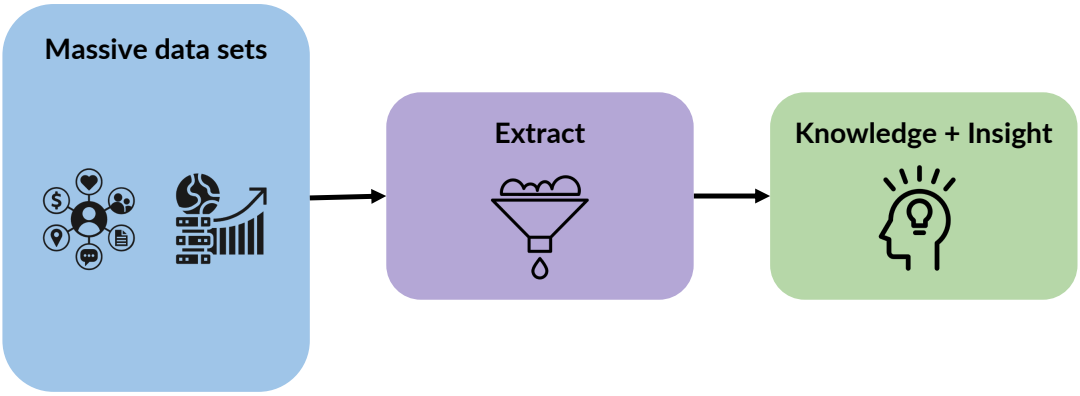
Practical Data Science

Explore the Use Case and Analyze the Dataset

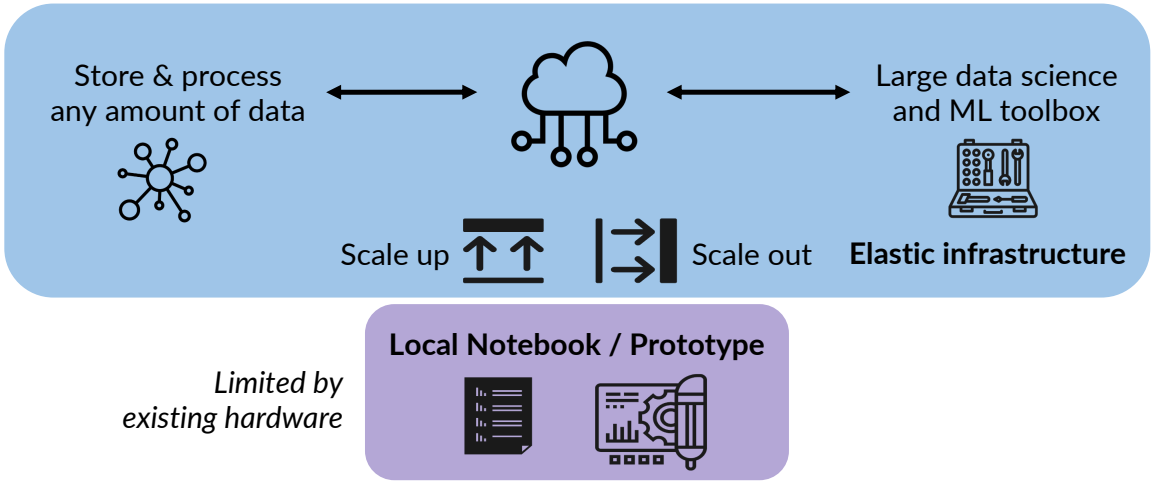
AI, ML, DL, data science...?



Practical data science



Practical data science in the cloud



Data science and ML toolbox

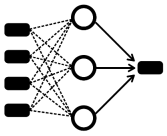
Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Use Case and Dataset

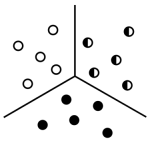
Introduction

Popular ML tasks and learning paradigms



Classification & Regression

Supervised



Clustering

Unsupervised

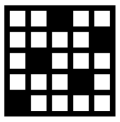


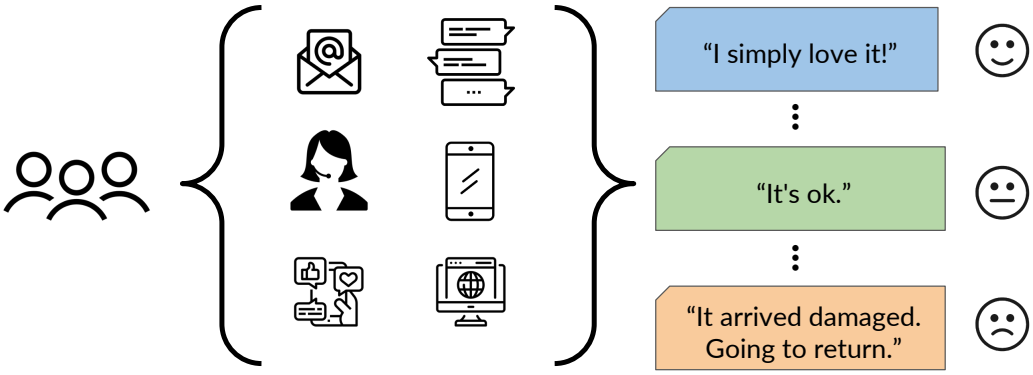
Image Processing

Computer Vision

Text Analysis

NLP / NLU

Multi-class classification for sentiment analysis of product reviews

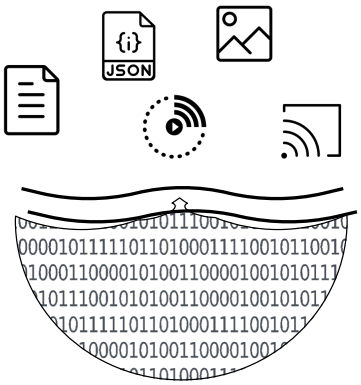


Working with product reviews data

Input feature for model training	Label for model training
Review Text	**Sentiment**
I simply love it!	1 (positive)
It's ok.	0 (neutral)
It arrived damaged, going to return	-1 (negative)

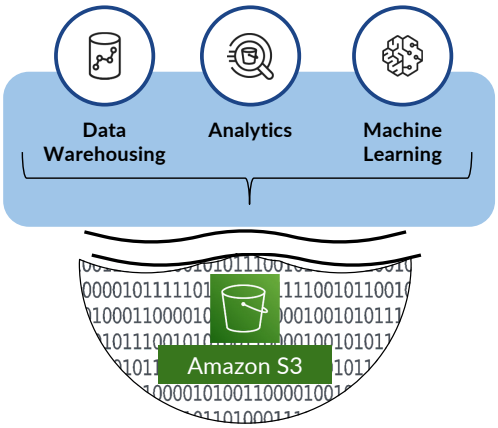
Data Ingestion & Exploration

Ingest data into data lakes



- Centralized and secure repository
- Store, discover and share data at any scale
  - structured relational data
  - semi-structured data
  - unstructured data
  - streaming data
- Governance

Data lakes on Amazon S3



- Amazon Simple Storage Service (Amazon S3)
- Object storage
- Durable, available, exabyte scale
- Secure, compliant, auditable

AWS Data Wrangler

- Open source Python library
- Connects pandas DataFrames and AWS data services
- Load/unload data from
  - data lakes
  - data warehouses
  - databases

```
!pip install awswrangler

import awswrangler as wr
import pandas as pd

# Retrieving the data directly from Amazon
S3
df = wr.s3.read_csv(
          path='s3://bucket/prefix/')
```


Register data with AWS Glue Data Catalog

AWS Glue
Data Catalog

Name	reviews
Database	dsoaws_deep_learning
Classification	csv
Location	s3://<bucket>/<prefix>

- Creates reference to data ("S3-to-table" mapping)
- Just metadata / schema stored in tables
- No data is moved
- *AWS Glue Crawlers* can be set up to automatically
  - infer data schema
  - update data catalog

Register data with AWS Glue Data Catalog

AWS Glue
Data Catalog

Name	reviews
Database	dsoaws_deep_learning
Classification	csv
Location	s3://<bucket>/<prefix>

```
import awswrangler as wr

# Create a database in the
# AWS Glue Data Catalog
wr.catalog.create_database(
         name=...)

# Create CSV table (metadata only) in the
# AWS Glue Data Catalog
wr.catalog.create_csv_table(
         table=...,
         column_types=...,
     ...)
```


Query data with Amazon Athena

Amazon
Athena

- Query data in S3
- Using SQL
- No infrastructure to set up
- Schema lookup in AWS Glue Data Catalog
- No data to load

Python

```
import awswrangler as wr

# Create Amazon Athena S3 bucket
wr.athena.create_athena_bucket()

# Execute SQL query on Amazon Athena
df = wr.athena.read_sql_query(
    sql=...,
    database=...)
```

SQL

```
'SELECT product_category FROM reviews'
```


Query data with Amazon Athena

SQL

presto

- Complex analytical queries
- Gigabytes > Terabytes > Petabytes
- Scales automatically
- Runs queries in parallel
- Based on Presto
- No infrastructure setup / no data movement required

Data Visualization

Popular Python data analysis & visualization tools

pandas

```
pip install pandas
```

NumPy

```
pip install numpy
```

matplotlib

```
pip install matplotlib
```

seaborn

```
pip install seaborn
```


How many reviews are in each *sentiment class*?

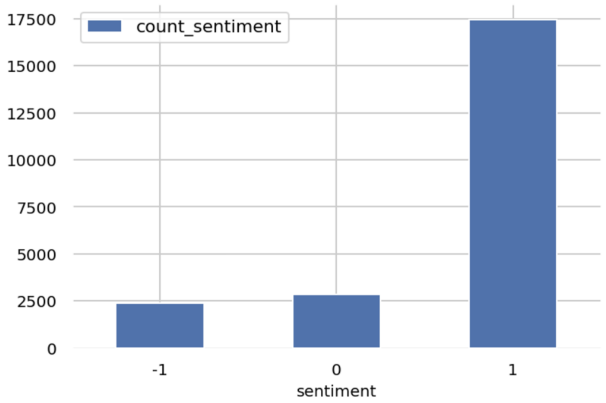
SQL Query

```
SELECT sentiment, COUNT(*) AS count_sentiment
FROM dsoaws_deep_learning.reviews
GROUP BY sentiment
ORDER BY sentiment DESC, count_sentiment
```

Python visualization code

```
import matplotlib.pyplot as plt
chart = df.plot.bar(
          x="sentiment",
     y="count_sentiment")

plt.xlabel("sentiment")
plt.show(chart)
```

How many reviews are in each *sentiment class*?



What is the distribution of review lengths? *(number of words)*

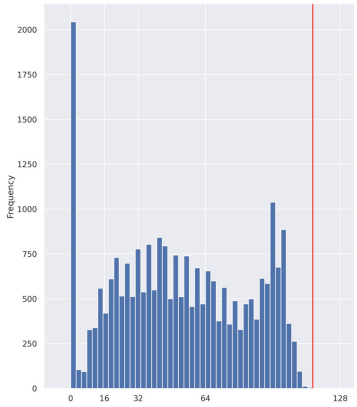
SQL Query

```
SELECT CARDINALITY(SPLIT(review_body, ' ')) as num_words
FROM dsoaws_deep_learning.reviews
```

Python visualization code

```
summary = df["num_words"].describe(
     percentiles=[0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00])

df["num_words"].plot.hist(
     xticks=[0, 16, 32, 64, 128, 256], bins=100,
     range=[0, 256]).axvline(x=summary["100%"], c="red")
```


What is the distribution of review lengths? *(number of words)*



mean	52.51
std	31.38
min	1.00
10%	10.00
20%	22.00
30%	32.00
40%	41.00
50%	51.00
60%	61.00
70%	73.00
80%	88.00
90%	97.00
100%	**115.00**

Register and visualize dataset

Introduction

In this lab you will ingest and transform the customer product reviews dataset. Then you will use AWS data stack services such as AWS Glue and Amazon Athena for ingesting and querying the dataset. Finally you will use AWS Data Wrangler to analyze the dataset and plot some visuals extracting insights.

Table of Contents

- 1. Ingest and transform the public dataset
  - 1.1. List the dataset files in the public S3 bucket
    - Exercise 1
  - 1.2. Copy the data locally to the notebook
  - 1.3. Transform the data
  - 1.4 Write the data to a CSV file
- 2. Register the public dataset for querying and visualizing
  - 2.1. Register S3 dataset files as a table for querying
    - Exercise 2
  - 2.2. Create default S3 bucket for Amazon Athena
- 3. Visualize data
  - 3.1. Preparation for data visualization
  - 3.2. How many reviews per sentiment?
    - Exercise 3
  - 3.3. Which product categories are highest rated by average sentiment?
  - 3.4. Which product categories have the most reviews?
    - Exercise 4
  - 3.5. What is the breakdown of sentiments per product category?
  - 3.6. Analyze the distribution of review word counts

Let's install the required modules first.

In [1]:
```
# please ignore warning messages during the installation
!pip install --disable-pip-version-check -q sagemaker==2.35.0
!pip install --disable-pip-version-check -q pandas==1.1.4
!pip install --disable-pip-version-check -q awswrangler==2.7.0
!pip install --disable-pip-version-check -q numpy==1.18.5
!pip install --disable-pip-version-check -q seaborn==0.11.0
!pip install --disable-pip-version-check -q matplotlib===3.3.3
```

```
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
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
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
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
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
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
```

1. Ingest and transform the public dataset

The dataset Women's Clothing Reviews has been chosen as the main dataset.

It is shared in a public Amazon S3 bucket, and is available as a comma-separated value (CSV) text format:

`s3://dlai-practical-data-science/data/raw/womens_clothing_ecommerce_reviews.csv`

1.1. List the dataset files in the public S3 bucket

The AWS Command Line Interface (CLI) is a unified tool to manage your AWS services. With just one tool, you can control multiple AWS services from the command line and automate them through scripts. You will use it to list the dataset files.

View dataset files in CSV format

`aws s3 ls [bucket_name]` function lists all objects in the S3 bucket. Let's use it to view the reviews data files in CSV format:

Exercise 1

View the list of the files available in the public bucket `s3://dlai-practical-data-science/data/raw/`.

Instructions: Use `aws s3 ls [bucket_name]` function. To run the AWS CLI command from the notebook you will need to put an exclamation mark in front of it: `!aws`. You should see the data file `womens_clothing_ecommerce_reviews.csv` in the list.

In [2]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
None # Replace None
### END SOLUTION - DO NOT delete this comment for grading purposes
!aws s3 ls s3://dlai-practical-data-science/data/raw/
# EXPECTED OUTPUT
# ... womens_clothing_ecommerce_reviews.csv
```

```
2021-04-30 02:21:06    8457214 womens_clothing_ecommerce_reviews.csv
```

1.2. Copy the data locally to the notebook

`aws s3 cp [bucket_name/file_name] [file_name]` function copies the file from the S3 bucket into the local environment or into another S3 bucket. Let's use it to copy the file with the dataset locally.

In [3]:
```
!aws s3 cp s3://dlai-practical-data-science/data/raw/womens_clothing_ecom
```

```
download: s3://dlai-practical-data-science/data/raw/womens_clothing_ecomm
erce_reviews.csv to ./womens_clothing_ecommerce_reviews.csv
```

Now use the Pandas dataframe to load and preview the data.

In [4]:
```
import pandas as pd
import csv

df = pd.read_csv('./womens_clothing_ecommerce_reviews.csv',
                 index_col=0)

df.shape
```

Out[4]:
```
(23486, 10)
```

In [5]:
```
df.head()
```

Out[5]:

	Clothing ID	Age	Title	Review Text	Rating	Recommended IND	Positive Feedback Count	Division Name	Depa
0	847	33	Cute, crisp shirt	If this product was in petite i would get the...	4	1	2	General	
1	1080	34	NaN	Love this dress! it's sooo pretty. i happene...	5	1	4	General	D
2	1077	60	Some major design flaws	I had such high hopes for this dress and reall...	3	0	0	General	D
3	1049	50	My favorite buy!	I love love love this jumpsuit. it's fun fl...	5	1	0	General Petite	E
4	847	47	Flattering shirt	This shirt is very flattering to all due to th...	5	1	6	General	

1.3. Transform the data

To simplify the task, you will transform the data into a comma-separated value (CSV) file that contains only a `review_body`, `product_category`, and `sentiment` derived from the original data.

In [6]:

```
df_transformed = df.rename(columns={'Review Text': 'review_body',
                                    'Rating': 'star_rating',
                                    'Class Name': 'product_category'})
df_transformed.drop(columns=['Clothing ID', 'Age', 'Title', 'Recommended
                     inplace=True)

df_transformed.dropna(inplace=True)

df_transformed.shape
```

Out[6]:

```
(22628, 3)
```

Now convert the `star_rating` into the `sentiment` (positive, neutral, negative), which later on will be for the prediction.

In [7]:
```
def to_sentiment(star_rating):
    if star_rating in {1, 2}: # negative
        return -1
    if star_rating == 3:      # neutral
        return 0
    if star_rating in {4, 5}: # positive
        return 1

# transform star_rating into the sentiment
df_transformed['sentiment'] = df_transformed['star_rating'].apply(lambda
    to_sentiment(star_rating=star_rating)
)

# drop the star rating column
df_transformed.drop(columns=['star_rating'],
                    inplace=True)

# remove reviews for product_categories with < 10 reviews
df_transformed = df_transformed.groupby('product_category').filter(lambda

df_transformed.shape
```

Out[7]:
```
(22626, 3)
```

In [9]:
```
# preview the results
df_transformed
```

Out[9]:

	sentiment	review_body	product_category
0	1	If this product was in petite i would get the...	Blouses
1	1	Love this dress! it's sooo pretty. i happene...	Dresses
2	0	I had such high hopes for this dress and reall...	Dresses
3	1	I love love love this jumpsuit. it's fun fl...	Pants
4	1	This shirt is very flattering to all due to th...	Blouses
...	...	...	...
23481	1	I was very happy to snag this dress at such a ...	Dresses
23482	0	It reminds me of maternity clothes. soft stre...	Knits
23483	0	This fit well but the top was very see throug...	Dresses
23484	0	I bought this dress for a wedding i have this ...	Dresses
23485	1	This dress in a lovely platinum is feminine an...	Dresses

22626 rows × 3 columns

1.4 Write the data to a CSV file

In [10]:
```
df_transformed.to_csv('./womens_clothing_ecommerce_reviews_transformed.cs
                      index=False)
```

In [11]:
```
!head -n 5 ./womens_clothing_ecommerce_reviews_transformed.csv
```

```
sentiment,review_body,product_category
1,If this product was in petite  i would get the petite. the regular is a
little long on me but a tailor can do a simple fix on that.     fits nice
ly! i'm 5'4  130lb and pregnant so i bough t medium to grow into.     the
tie can be front or back so provides for some nice flexibility on form fi
tting.,Blouses
1,"Love this dress!  it's sooo pretty.  i happened to find it in a store
and i'm glad i did bc i never would have ordered it online bc it's petite
.  i bought a petite and am 5'8"".  i love the length on me- hits just a
little below the knee.  would definitely be a true midi on someone who is
truly petite.",Dresses
0,I had such high hopes for this dress and really wanted it to work for m
e. i initially ordered the petite small (my usual size) but i found this
to be outrageously small. so small in fact that i could not zip it up! i
reordered it in petite medium  which was just ok. overall  the top half w
as comfortable and fit nicely  but the bottom half had a very tight under
layer and several somewhat cheap (net) over layers. imo  a major design f
law was the net over layer sewn directly into the zipper - it c,Dresses
1,I love  love  love this jumpsuit. it's fun  flirty  and fabulous! every
time i wear it  i get nothing but great compliments!,Pants
```

2. Register the public dataset for querying and visualizing

You will register the public dataset into an S3-backed database table so you can query and visualize our dataset at scale.

2.1. Register S3 dataset files as a table for querying

Let's import required modules.

`boto3` is the AWS SDK for Python to create, configure, and manage AWS services, such as Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3). The SDK provides an object-oriented API as well as low-level access to AWS services.

`sagemaker` is the SageMaker Python SDK which provides several high-level abstractions for working with the Amazon SageMaker.

In [12]:
```
import boto3
import sagemaker
import pandas as pd
import numpy as np
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c1/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
account_id = sess.account_id

print('S3 Bucket: {}'.format(bucket))
print('Region: {}'.format(region))
print('Account ID: {}'.format(account_id))
```

```
S3 Bucket: sagemaker-us-east-1-407738478633
Region: us-east-1
Account ID: <bound method Session.account_id of <sagemaker.session.Sessio
n object at 0x7ff92b1e7190>>
```

Review the empty bucket which was created automatically for this account.

Instructions:

- open the link
- click on the S3 bucket name `sagemaker-us-east-1-ACCOUNT`
- check that it is empty at this stage

In [13]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="top" href="https://s3.console.aws.amaz
```

Review Amazon S3 buckets

Copy the file into the S3 bucket.

In [14]:
```
!aws s3 cp ./womens_clothing_ecommerce_reviews_transformed.csv s3://$buck
```

```
upload: ./womens_clothing_ecommerce_reviews_transformed.csv to s3://sagem
aker-us-east-1-407738478633/data/transformed/womens_clothing_ecommerce_re
views_transformed.csv
```

Review the bucket with the file we uploaded above.

Instructions:

- open the link
- check that the CSV file is located in the S3 bucket
- check the location directory structure is the same as in the CLI command above
- click on the file name and see the available information about the file (region, size, S3 URI, Amazon Resource Name (ARN))

In [15]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="top" href="https://s3.console.aws.amaz
```

Review Amazon S3 buckets

Import AWS Data Wrangler

AWS Data Wrangler is an AWS Professional Service open source python initiative that extends the power of Pandas library to AWS connecting dataframes and AWS data related services (Amazon Redshift, AWS Glue, Amazon Athena, Amazon EMR, Amazon QuickSight, etc).

Built on top of other open-source projects like Pandas, Apache Arrow, Boto3, SQLAlchemy, Psycopg2 and PyMySQL, it offers abstracted functions to execute usual ETL tasks like load/unload data from data lakes, data warehouses and databases.

Review the AWS Data Wrangler documentation: https://aws-data-wrangler.readthedocs.io/en/stable/

In [16]:
```
import awswrangler as wr
```

Create AWS Glue Catalog database

The data catalog features of **AWS Glue** and the inbuilt integration to Amazon S3 simplify the process of identifying data and deriving the schema definition out of the discovered data. Using AWS Glue crawlers within your data catalog, you can traverse your data stored in Amazon S3 and build out the metadata tables that are defined in your data catalog.

Here you will use `wr.catalog.create_database` function to create a database with the name `dsoaws_deep_learning` ("dsoaws" stands for "Data Science on AWS").

In [17]:
```
wr.catalog.create_database(
    name='dsoaws_deep_learning',
    exist_ok=True
)
```

In [18]:
```
dbs = wr.catalog.get_databases()

for db in dbs:
    print("Database name: " + db['Name'])
```

```
Database name: dsoaws_deep_learning
```

Review the created database in the AWS Glue Catalog.

Instructions:

- open the link
- on the left side panel notice that you are in the AWS Glue -> Data Catalog -> Databases
- check that the database `dsoaws_deep_learning` has been created
- click on the name of the database
- click on the `Tables in dsoaws_deep_learning` link to see that there are no tables

In [19]:
```
from IPython.core.display import display, HTML

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

Review AWS Glue Databases

Register CSV data with AWS Glue Catalog

Exercise 2

Register CSV data with AWS Glue Catalog.

Instructions: Use `wr.catalog.create_csv_table` function with the following parameters

```
res = wr.catalog.create_csv_table(
    database='...', # AWS Glue Catalog database name
    path='s3://{}/data/transformed/'.format(bucket), # S3 object
path for the data
    table='reviews', # registered table name
    columns_types={
        'sentiment': 'int',
        'review_body': 'string',
        'product_category': 'string'
    },
    mode='overwrite',
    skip_header_line_count=1,
    sep=','
)
```

In [22]:
```
wr.catalog.create_csv_table(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    database= 'dsoaws_deep_learning', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    path='s3://{}/data/transformed/'.format(bucket),
    table="reviews",
    columns_types={
        'sentiment': 'int',
        'review_body': 'string',
        'product_category': 'string'
    },
    mode='overwrite',
    skip_header_line_count=1,
    sep=','
)
```

Review the registered table in the AWS Glue Catalog.

Instructions:

- open the link
- on the left side panel notice that you are in the AWS Glue -> Data Catalog -> Databases -> Tables
- check that you can see the table `reviews` from the database `dsoaws_deep_learning` in the list
- click on the name of the table
- explore the available information about the table (name, database, classification, location, schema etc.)

In [23]:
```
from IPython.core.display import display, HTML

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

Review AWS Glue Catalog

Review the table shape:

In [24]:
```
table = wr.catalog.table(database='dsoaws_deep_learning',
                         table='reviews')
table
```

Out[24]:

	Column Name	Type	Partition	Comment
0	sentiment	int	False	
1	review_body	string	False	
2	product_category	string	False	

2.2. Create default S3 bucket for Amazon Athena

Amazon Athena requires this S3 bucket to store temporary query results and improve performance of subsequent queries.

The contents of this bucket are mostly binary and human-unreadable.

In [25]:
```
# S3 bucket name
wr.athena.create_athena_bucket()

# EXPECTED OUTPUT
# 's3://aws-athena-query-results-ACCOUNT-REGION/'
```

Out[25]:
```
's3://aws-athena-query-results-407738478633-us-east-1/'
```

3. Visualize data

Reviews dataset - column descriptions

- `sentiment` : The review's sentiment (-1, 0, 1).
- `product_category` : Broad product category that can be used to group reviews (in this case digital videos).
- `review_body` : The text of the review.

3.1. Preparation for data visualization

Imports

In [26]:
```
import numpy as np
import seaborn as sns

import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

Settings

Set AWS Glue database and table name.

In [27]:
```
# Do not change the database and table names - they are used for grading
database_name = 'dsoaws_deep_learning'
table_name = 'reviews'
```

Set seaborn parameters. You can review seaborn documentation following the link.

In [28]:
```
sns.set_style = 'seaborn-whitegrid'

sns.set(rc={"font.style":"normal",
            "axes.facecolor":"white",
            'grid.color': '.8',
            'grid.linestyle': '-',
            "figure.facecolor":"white",
            "figure.titlesize":20,
            "text.color":"black",
            "xtick.color":"black",
            "ytick.color":"black",
            "axes.labelcolor":"black",
            "axes.grid":True,
            'axes.labelsize':10,
            'xtick.labelsize':10,
            'font.size':10,
            'ytick.labelsize':10})
```

Helper code to display values on barplots:

Run SQL queries using Amazon Athena

Amazon Athena lets you query data in Amazon S3 using a standard SQL interface. It reflects the databases and tables in the AWS Glue Catalog. You can create interactive queries and perform any data manipulations required for further downstream processing.

Standard SQL query can be saved as a string and then passed as a parameter into the Athena query. Run the following cells as an example to count the total number of reviews by sentiment. The SQL query here will take the following form:

```
SELECT column_name, COUNT(column_name) as new_column_name
FROM table_name
GROUP BY column_name
ORDER BY column_name
```

If you are not familiar with the SQL query statements, you can review some tutorials following the link.

3.2. How many reviews per sentiment?

Set the SQL statement to find the count of sentiments:

In [35]:
```
statement_count_by_sentiment = '''

SELECT sentiment, COUNT(sentiment) AS count_sentiment
FROM reviews
GROUP BY sentiment
ORDER BY sentiment
'''

print(statement_count_by_sentiment)
```

```
SELECT sentiment, COUNT(sentiment) AS count_sentiment
FROM reviews
GROUP BY sentiment
ORDER BY sentiment
```

Query data in Amazon Athena database cluster using the prepared SQL statement:

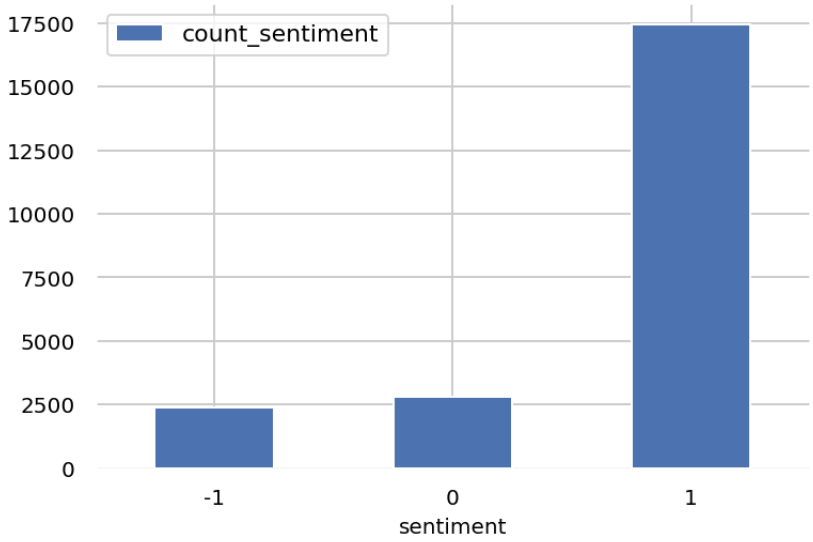
In [36]:
```
df_count_by_sentiment = wr.athena.read_sql_query(
    sql=statement_count_by_sentiment,
    database=database_name
)

print(df_count_by_sentiment)
```

```
   sentiment  count_sentiment
0         -1             2370
1          0             2823
2          1            17433
```

Preview the results of the query:

In [37]:
```
df_count_by_sentiment.plot(kind='bar', x='sentiment', y='count_sentiment'
```

Out[37]:
```
<AxesSubplot:xlabel='sentiment'>
```



Exercise 3

Use Amazon Athena query with the standard SQL statement passed as a parameter, to calculate the total number of reviews per `product_category` in the table `reviews`.

Instructions: Pass the SQL statement of the form

```
SELECT category_column, COUNT(column_name) AS new_column_name
FROM table_name
GROUP BY category_column
ORDER BY new_column_name DESC
```

as a triple quote string into the variable `statement_count_by_category`. Please use the column `sentiment` in the `COUNT` function and give it a new name `count_sentiment`.

In [39]:

```
# Replace all None
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
statement_count_by_category = """
SELECT product_category, COUNT(sentiment) AS count_sentiment
FROM reviews
GROUP BY product_category
ORDER BY product_category DESC
"""
### END SOLUTION - DO NOT delete this comment for grading purposes
print(statement_count_by_category)
```

```
SELECT product_category, COUNT(sentiment) AS count_sentiment
FROM reviews
GROUP BY product_category
ORDER BY product_category DESC
```

Query data in Amazon Athena database passing the prepared SQL statement:

In [40]:

```
%%time
df_count_by_category = wr.athena.read_sql_query(
    sql=statement_count_by_category,
    database=database_name
)

df_count_by_category

# EXPECTED OUTPUT
# Dresses: 6145
# Knits: 4626
# Blouses: 2983
# Sweaters: 1380
# Pants: 1350
# ...
```

```
CPU times: user 386 ms, sys: 28.7 ms, total: 415 ms
Wall time: 3.65 s
```

Out[40]:

	product_category	count_sentiment
0	Trend	118
1	Swim	332
2	Sweaters	1380
3	Sleep	214
4	Skirts	903
5	Shorts	304
6	Pants	1350
7	Outerwear	319
8	Lounge	669
9	Legwear	158
10	Layering	132
11	Knits	4626
12	Jeans	1104
13	Jackets	683
14	Intimates	147
15	Fine gauge	1059
16	Dresses	6145
17	Blouses	2983

3.3. Which product categories are highest rated by average sentiment?

Set the SQL statement to find the average sentiment per product category, showing the results in the descending order:

In [41]:

```
statement_avg_by_category = """
SELECT product_category, AVG(sentiment) AS avg_sentiment
FROM {}
GROUP BY product_category
ORDER BY avg_sentiment DESC
""".format(table_name)

print(statement_avg_by_category)
```

```
SELECT product_category, AVG(sentiment) AS avg_sentiment
FROM reviews
GROUP BY product_category
ORDER BY avg_sentiment DESC
```

Query data in Amazon Athena database passing the prepared SQL statement:

In [42]:
```
%%time
df_avg_by_category = wr.athena.read_sql_query(
    sql=statement_avg_by_category,
    database=database_name
)
```

```
CPU times: user 330 ms, sys: 17.2 ms, total: 348 ms
Wall time: 4.83 s
```

Preview the query results in the temporary S3 bucket: `s3://aws-athena-query-results-ACCOUNT-REGION/`

Instructions:

- open the link
- check the name of the S3 bucket
- briefly check the content of it

In [43]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="top" href="https://s3.console.aws.amaz
```

Review Amazon S3 buckets

Preview the results of the query:

In [44]:
```
df_avg_by_category
```

Out[44]:

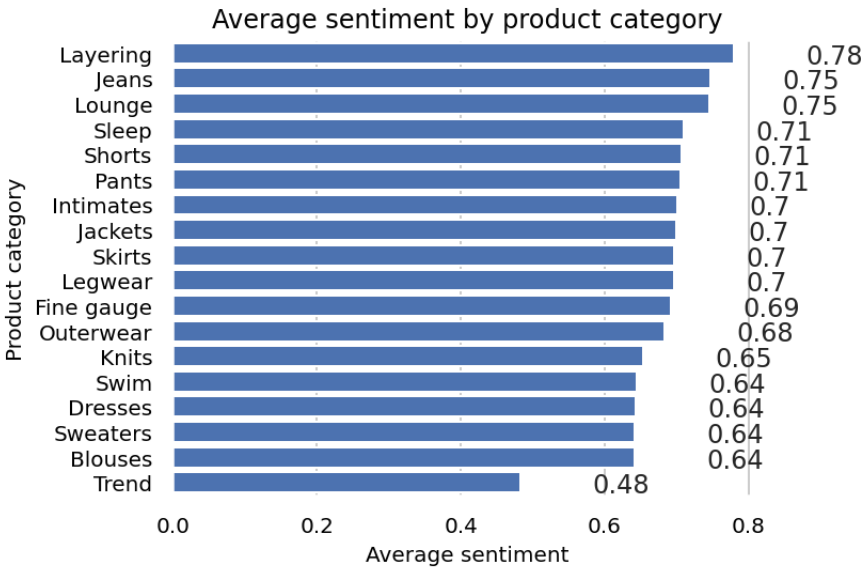
	product_category	avg_sentiment
0	Layering	0.780303
1	Jeans	0.746377
2	Lounge	0.745889
3	Sleep	0.710280
4	Shorts	0.707237
5	Pants	0.705185
6	Intimates	0.700680
7	Jackets	0.699854
8	Skirts	0.696567
9	Legwear	0.696203
10	Fine gauge	0.692162
11	Outerwear	0.683386
12	Knits	0.653913
13	Swim	0.644578
14	Dresses	0.643287
15	Sweaters	0.641304
16	Blouses	0.641301
17	Trend	0.483051

Visualization

In [45]:

```
def show_values_barplot(axs, space):
    def _show_on_plot(ax):
        for p in ax.patches:
            _x = p.get_x() + p.get_width() + float(space)
            _y = p.get_y() + p.get_height()
            value = round(float(p.get_width()),2)
            ax.text(_x, _y, value, ha="left")

    if isinstance(axs, np.ndarray):
        for idx, ax in np.ndenumerate(axs):
            _show_on_plot(ax)
    else:
        _show_on_plot(axs)
```

In [46]:
```
# Create plot
barplot = sns.barplot(
    data = df_avg_by_category,
    y='product_category',
    x='avg_sentiment',
    color="b",
    saturation=1
)

# Set the size of the figure
sns.set(rc={'figure.figsize':(15.0, 10.0)})

# Set title and x-axis ticks
plt.title('Average sentiment by product category')
#plt.xticks([-1, 0, 1], ['Negative', 'Neutral', 'Positive'])

# Helper code to show actual values afters bars
show_values_barplot(barplot, 0.1)

plt.xlabel("Average sentiment")
plt.ylabel("Product category")

plt.tight_layout()
# Do not change the figure name - it is used for grading purposes!
plt.savefig('avg_sentiment_per_category.png', dpi=300)

# Show graphic
plt.show(barplot)
```



In [47]:
```
# Upload image to S3 bucket
sess.upload_data(path='avg_sentiment_per_category.png', bucket=bucket, ke
```

Out[47]:
```
's3://sagemaker-us-east-1-407738478633/images/avg_sentiment_per_category.
png'
```

Review the bucket on the account.

Instructions:

- open the link
- click on the S3 bucket name `sagemaker-us-east-1-ACCOUNT`
- open the images folder
- check the existence of the image `avg_sentiment_per_category.png`
- if you click on the image name, you can see the information about the image file. You can also download the file with the command on the top right Object Actions -> Download / Download as

In [48]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="top" href="https://s3.console.aws.amaz
```

Review Amazon S3 buckets

3.4. Which product categories have the most reviews?

Set the SQL statement to find the count of sentiment per product category, showing the results in the descending order:

In [53]:
```
statement_count_by_category_desc = """
SELECT product_category, COUNT(*) AS count_reviews
FROM {}
GROUP BY product_category
ORDER BY count_reviews DESC
""".format(table_name)

print(statement_count_by_category_desc)
```

```
SELECT product_category, COUNT(*) AS count_reviews
FROM reviews
GROUP BY product_category
ORDER BY count_reviews DESC
```

Query data in Amazon Athena database passing the prepared SQL statement:

In [54]:
```
%%time
df_count_by_category_desc = wr.athena.read_sql_query(
    sql=statement_count_by_category_desc,
    database=database_name
)
```

```
CPU times: user 276 ms, sys: 11 ms, total: 287 ms
Wall time: 3.58 s
```

Store maximum number of sentiment for the visualization plot:

In [55]:
```
max_sentiment = df_count_by_category_desc['count_reviews'].max()
print('Highest number of reviews (in a single category): {}'.format(max_s
```

```
Highest number of reviews (in a single category): 6145
```

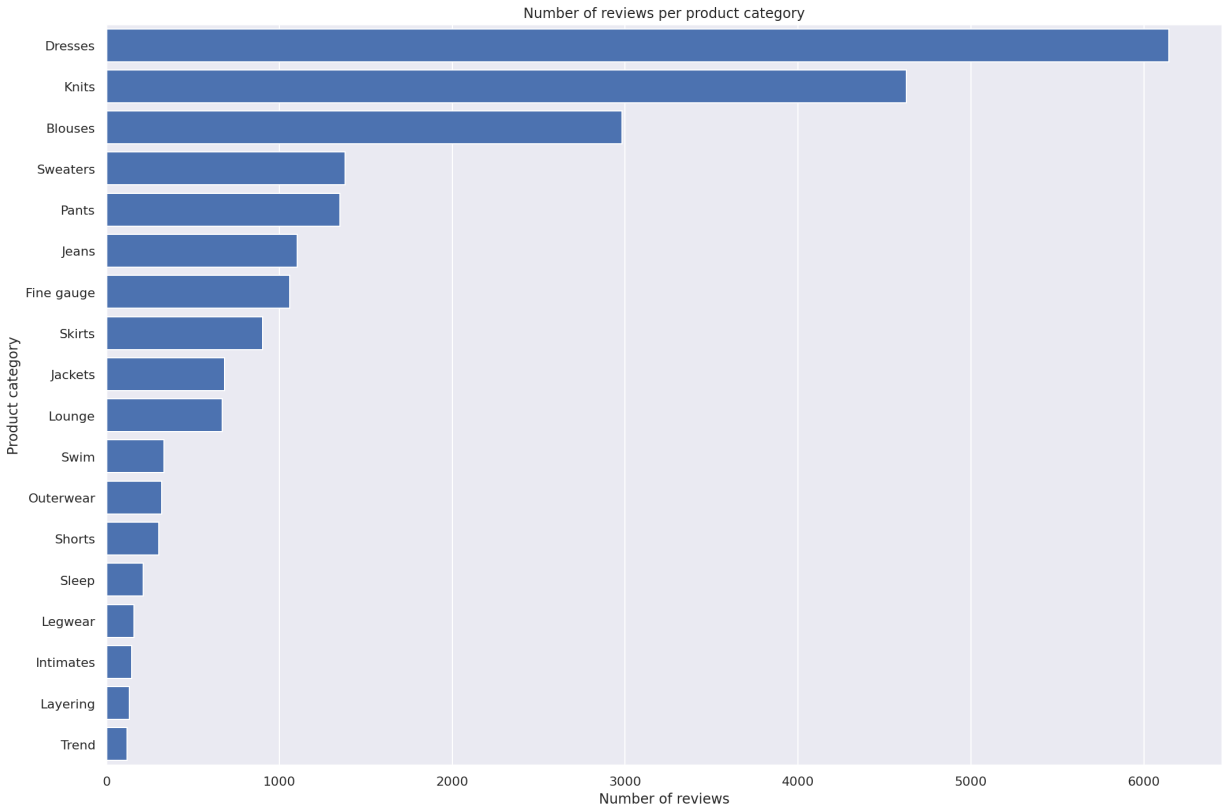
Visualization

Exercise 4

Use `barplot` function to plot number of reviews per product category.

Instructions: Use the `barplot` chart example in the previous section, passing the newly defined dataframe `df_count_by_category_desc` with the count of reviews. Here, please put the `product_category` column into the `y` argument.

In [58]:
```
# Create seaborn barplot
barplot = sns.barplot(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    data=df_count_by_category_desc, # Replace None
    y= 'product_category', # Replace None
    x= 'count_reviews', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    color="b",
    saturation=1
)

# Set the size of the figure
sns.set(rc={'figure.figsize':(15.0, 10.0)})

# Set title
plt.title("Number of reviews per product category")
plt.xlabel("Number of reviews")
plt.ylabel("Product category")

plt.tight_layout()

# Do not change the figure name - it is used for grading purposes!
plt.savefig('num_reviews_per_category.png', dpi=300)

# Show the barplot
plt.show(barplot)
```



In [59]:
```
# Upload image to S3 bucket
sess.upload_data(path='num_reviews_per_category.png', bucket=bucket, key_
```

Out[59]:
```
's3://sagemaker-us-east-1-407738478633/images/num_reviews_per_category.pn
g'
```

3.5. What is the breakdown of sentiments per product category?

Set the SQL statement to find the count of sentiment per product category and sentiment:

In [60]:
```
statement_count_by_category_and_sentiment = """
SELECT product_category,
         sentiment,
         COUNT(*) AS count_reviews
FROM {}
GROUP BY  product_category, sentiment
ORDER BY  product_category ASC, sentiment DESC, count_reviews
""".format(table_name)

print(statement_count_by_category_and_sentiment)
```

```
SELECT product_category,
         sentiment,
         COUNT(*) AS count_reviews
FROM reviews
GROUP BY  product_category, sentiment
ORDER BY  product_category ASC, sentiment DESC, count_reviews
```

Query data in Amazon Athena database passing the prepared SQL statement:

In [61]:
```
%%time
df_count_by_category_and_sentiment = wr.athena.read_sql_query(
    sql=statement_count_by_category_and_sentiment,
    database=database_name
)
```

```
CPU times: user 257 ms, sys: 22.9 ms, total: 280 ms
Wall time: 3.28 s
```

Prepare for stacked percentage horizontal bar plot showing proportion of sentiments per product category.

In [62]:
```
# Create grouped dataframes by category and by sentiment
grouped_category = df_count_by_category_and_sentiment.groupby('product_ca
grouped_star = df_count_by_category_and_sentiment.groupby('sentiment')

# Create sum of sentiments per star sentiment
df_sum = df_count_by_category_and_sentiment.groupby(['sentiment']).sum()

# Calculate total number of sentiments
total = df_sum['count_reviews'].sum()
print('Total number of reviews: {}'.format(total))
```

```
Total number of reviews: 22626
```

Create dictionary of product categories and array of star rating distribution per category.

In [63]:
```
distribution = {}
count_reviews_per_star = []
i=0

for category, sentiments in grouped_category:
    count_reviews_per_star = []
    for star in sentiments['sentiment']:
        count_reviews_per_star.append(sentiments.at[i, 'count_reviews'])
        i=i+1;
    distribution[category] = count_reviews_per_star
```

Build array per star across all categories.

In [64]:
```
distribution
```

Out[64]:
```
{'Blouses': [2256, 384, 343],
 'Dresses': [4634, 830, 681],
 'Fine gauge': [837, 118, 104],
 'Intimates': [117, 16, 14],
 'Jackets': [550, 61, 72],
 'Jeans': [909, 110, 85],
 'Knits': [3523, 605, 498],
 'Layering': [113, 9, 10],
 'Legwear': [126, 16, 16],
 'Lounge': [545, 78, 46],
 'Outerwear': [254, 29, 36],
 'Pants': [1074, 154, 122],
 'Shorts': [240, 39, 25],
 'Skirts': [714, 104, 85],
 'Sleep': [175, 16, 23],
 'Sweaters': [1036, 193, 151],
 'Swim': [252, 42, 38],
 'Trend': [78, 19, 21]}
```

In [65]:
```
df_distribution_pct = pd.DataFrame(distribution).transpose().apply(
    lambda num_sentiments: num_sentiments/sum(num_sentiments)*100, axis=1
)
df_distribution_pct.columns=['1', '0', '-1']
df_distribution_pct
```

Out[65]:

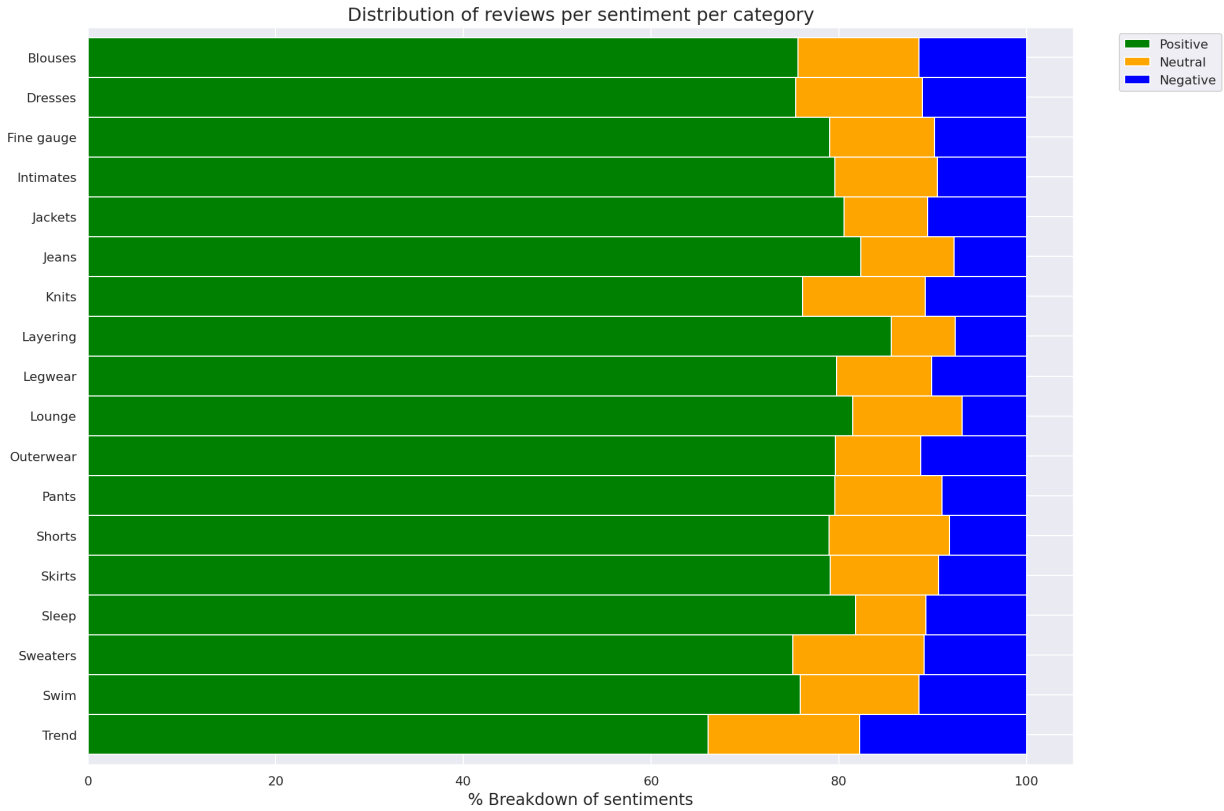
	1	0	-1
Blouses	75.628562	12.872947	11.498491
Dresses	75.410903	13.506916	11.082181
Fine gauge	79.036827	11.142587	9.820585
Intimates	79.591837	10.884354	9.523810
Jackets	80.527086	8.931186	10.541728
Jeans	82.336957	9.963768	7.699275
Knits	76.156507	13.078253	10.765240
Layering	85.606061	6.818182	7.575758
Legwear	79.746835	10.126582	10.126582
Lounge	81.464873	11.659193	6.875934
Outerwear	79.623824	9.090909	11.285266
Pants	79.555556	11.407407	9.037037
Shorts	78.947368	12.828947	8.223684
Skirts	79.069767	11.517165	9.413068
Sleep	81.775701	7.476636	10.747664
Sweaters	75.072464	13.985507	10.942029
Swim	75.903614	12.650602	11.445783
Trend	66.101695	16.101695	17.796610

Visualization

Plot the distributions of sentiments per product category.

In [66]:
```
categories = df_distribution_pct.index

# Plot bars
plt.figure(figsize=(10,5))

df_distribution_pct.plot(kind="barh",
                         stacked=True,
                         edgecolor='white',
                         width=1.0,
                         color=['green',
                                'orange',
                                'blue'])

plt.title("Distribution of reviews per sentiment per category",
          fontsize='16')

plt.legend(bbox_to_anchor=(1.04,1),
           loc="upper left",
           labels=['Positive',
                   'Neutral',
                   'Negative'])

plt.xlabel("% Breakdown of sentiments", fontsize='14')
plt.gca().invert_yaxis()
plt.tight_layout()

# Do not change the figure name - it is used for grading purposes!
plt.savefig('distribution_sentiment_per_category.png', dpi=300)
plt.show()
```

```
<Figure size 720x360 with 0 Axes>
```



In [67]:
```
# Upload image to S3 bucket
sess.upload_data(path='distribution_sentiment_per_category.png', bucket=b
```

Out[67]: 's3://sagemaker-us-east-1-407738478633/images/distribution_sentiment_per_category.png'

3.6. Analyze the distribution of review word counts

Set the SQL statement to count the number of the words in each of the reviews:

In [68]:
```
statement_num_words = """
    SELECT CARDINALITY(SPLIT(review_body, ' ')) as num_words
    FROM {}
""".format(table_name)

print(statement_num_words)
```

```
    SELECT CARDINALITY(SPLIT(review_body, ' ')) as num_words
    FROM reviews
```

Query data in Amazon Athena database passing the SQL statement:

In [69]:
```
%%time
df_num_words = wr.athena.read_sql_query(
    sql=statement_num_words,
    database=database_name
)
```

```
CPU times: user 423 ms, sys: 9.3 ms, total: 433 ms
Wall time: 3.76 s
```

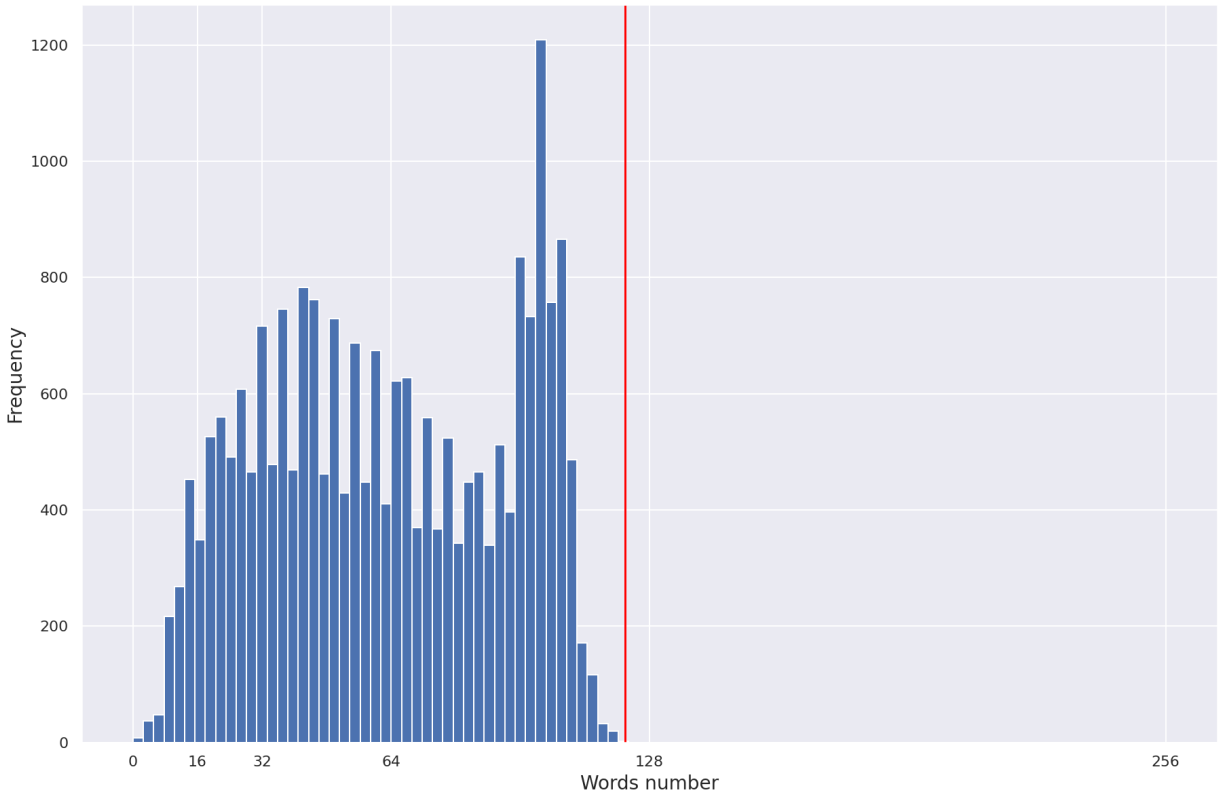
Print out and analyse some descriptive statistics:

In [70]:
```
summary = df_num_words["num_words"].describe(percentiles=[0.10, 0.20, 0.3
summary
```

Out[70]:
```
count    22626.000000
mean        62.709847
std         29.993735
min          2.000000
10%         22.000000
20%         33.000000
30%         42.000000
40%         51.000000
50%         61.000000
60%         72.000000
70%         86.000000
80%         97.000000
90%        103.000000
100%       122.000000
max        122.000000
Name: num_words, dtype: float64
```

Plot the distribution of the words number per review:

In [71]:
```
df_num_words["num_words"].plot.hist(xticks=[0, 16, 32, 64, 128, 256], bin
    x=summary["100%"], c="red"
)

plt.xlabel("Words number", fontsize='14')
plt.ylabel("Frequency", fontsize='14')
plt.savefig('distribution_num_words_per_review.png', dpi=300)
plt.show()
```



In [72]:
```
# Upload image to S3 bucket
sess.upload_data(path='distribution_num_words_per_review.png', bucket=buc
```

Out[72]:
```
's3://sagemaker-us-east-1-407738478633/images/distribution_num_words_per_
review.png'
```

Upload the notebook into S3 bucket for grading purposes.

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

In [73]:
```
!aws s3 cp ./C1_W1_Assignment.ipynb s3://$bucket/C1_W1_Assignment_Learner
```

```
upload: ./C1_W1_Assignment.ipynb to s3://sagemaker-us-east-1-407738478633
/C1_W1_Assignment_Learner.ipynb
```

Please go to the main lab window and click on `Submit` button (see the `Finish the lab` section of the instructions).

In [ ]:

DeepLearning.AI

aws

Practical Data Science

Statistical Bias and Feature Importance

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Statistical Bias

Statistical Bias

- Training data does not comprehensively represent the problem space
- Some elements of a dataset are more heavily weighted or represented

Fraud Detection

Biased models

- Imbalances in product review dataset

Statistical Bias Causes

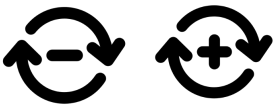
Statistical Bias – Causes

Activity Bias

Social Media Content

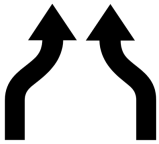
Societal Bias

Human Generated Content



Selection Bias

Feedback Loop

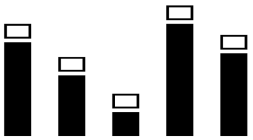


Data Drift

- *Covariant Drift*
- *Prior probability Drift*
- *Concept Drift*

Measuring Statistical Bias

Measure Imbalance in Data - Metrics



Class Imbalance (CI)

- Measures the imbalance in the number of members between different facet values.
- Does a ***product_category*** has disproportionately more reviews than others?

Measure Imbalance in Data - Metrics

Difference in Proportions of Labels (DPL)

- Measures the imbalance of positive outcomes between different facet values.
- Does a ***product_category*** has disproportionately higher ratings than others

Detecting Statistical Bias

- Amazon SageMaker Data Wrangler
- Amazon SageMaker Clarify

AWS Toolkit

Detect Statistical Bias - Amazon SageMaker Data Wrangler

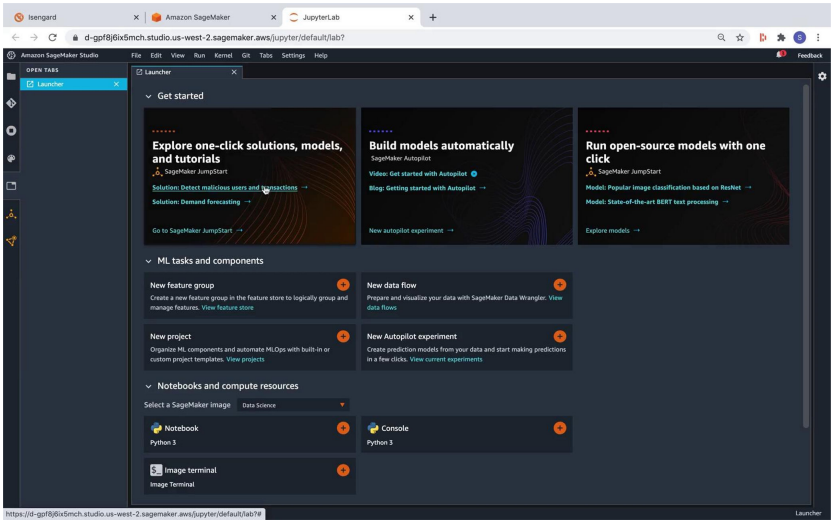
Source

Visualization

Transform

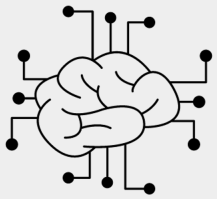
Statistical Bias Report

Feature Importance

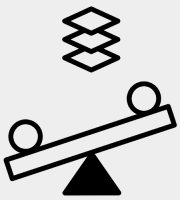


Detect Statistical Bias

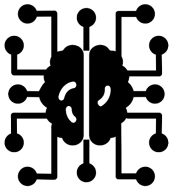
Amazon SageMaker Clarify



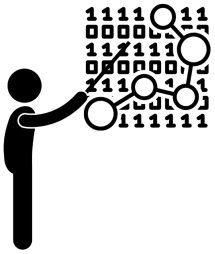
Detect Statistical Bias - Amazon SageMaker Clarify



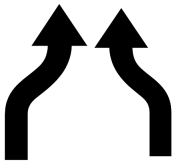
Statistical Bias Report



Model Bias Report



Explainability



Drift

Detect Statistical Bias - Amazon SageMaker Clarify

```
from sagemaker import clarify

clarify_processor = clarify.SageMakerClarifyProcessor(
    role=role,
        instance_count=1,
        instance_type='ml.c5.2xlarge'
        sagemaker_session=sess)

bias_report_output_path = << Define S3 path >>
```

Distributed cluster size

Type of each instance

S3 location to store bias report

Detect Statistical Bias - Amazon SageMaker Clarify

```
bias_data_config = clarify.DataConfig(
        s3_data_input_path=...,
        s3_output_path=...,
        label='sentiment',
        headers=df_balanced.columns.to_list(),
        dataset_type='text/csv')
```

Data Configuration

Detect Statistical Bias - Amazon SageMaker Clarify

```
bias_config = clarify.BiasConfig(
    label_values_or_threshold=[...],
    facet_name='product_category')
```

Bias Configuration

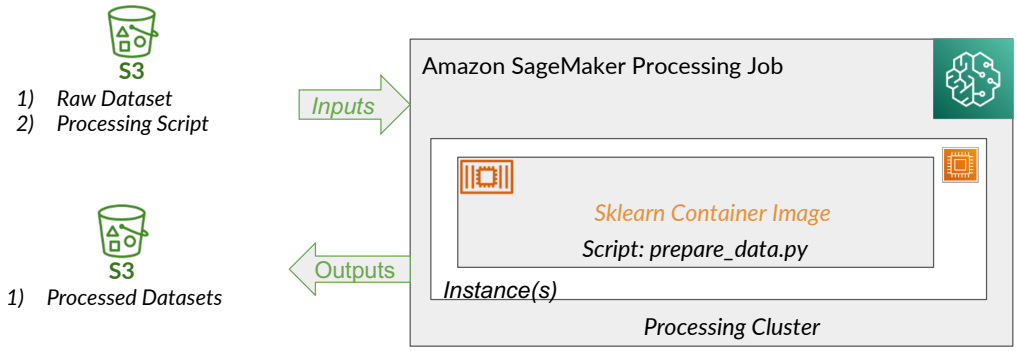
Detect Statistical Bias - Amazon SageMaker Clarify

```
clarify_processor.run_pre_training_bias(
    data_config=...,
    data_bias_config=...,
    methods=["CI", "DPL", ...],
    wait=<<False/True>>,
    logs=<<False/True>>)
```

Pre training bias job

Amazon SageMaker Processing

Execute preprocessing, post processing, model evaluation



Detect Statistical Bias - Amazon SageMaker Clarify

```
clarify_processor.run_pre_training_bias(
    data_config=...,
    data_bias_config=...,
    methods=["CI", "DPL", ...],
    wait=<<False/True>>,
    logs=<<False/True>>)
```

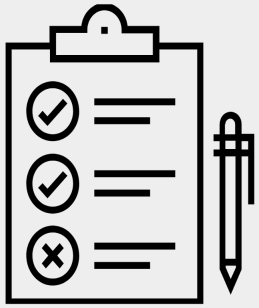
Result ?

Detecting Statistical Bias - Two Approaches

- Amazon SageMaker Data Wrangler
- Amazon SageMaker Clarify

Feature Importance

SHAP



Feature Importance

- Explains the features that make up the training data using a score (importance).
- How useful or valuable the feature is relative to other features.
- Predict the sentiment for a product → Which features play a role?

Feature Importance

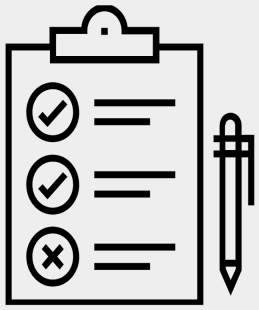
- **Open Source Framework - SHapley Additive exPlanations**

Feature Importance

- **Open Source Framework - SHAP**
  - Shapley values based on game theory.
  - Explain predictions of a ML model
    - Each feature value of training data instance is a player in a game
    - ML prediction is the payout
  - Local vs global explanations
  - SHAP can guarantee consistency and local accuracy.

Feature Importance

Amazon SageMaker Data Wrangler



Feature Importance - Amazon SageMaker Data Wrangler

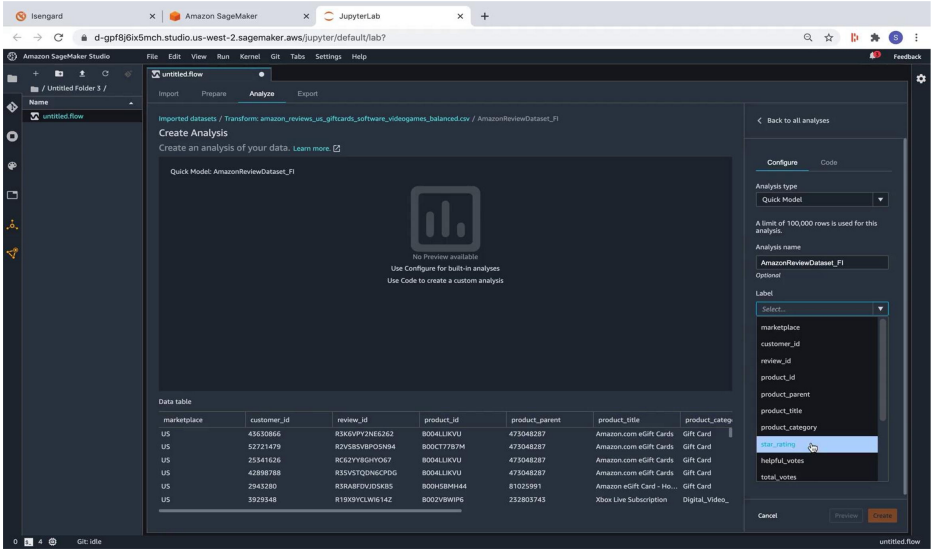
Source

Visualization

Transform

Bias Report

Feature Importance



DeepLearning.AI

aws

Detect data bias with Amazon SageMaker Clarify

Introduction

Bias can be present in your data before any model training occurs. Inspecting the dataset for bias can help detect collection gaps, inform your feature engineering, and understand societal biases the dataset may reflect. In this lab you will analyze bias on the dataset, generate and analyze bias report, and prepare the dataset for the model training.

Table of Contents

- 1. Analyze the dataset
  - 1.1. Create a pandas data frame from the CSV file
  - 1.2. Upload the dataset to S3 bucket
- 2. Analyze class imbalance on the dataset with Amazon SageMaker Clarify
  - 2.1. Configure a `DataConfig`
    - Exercise 1
  - 2.2. Configure `BiasConfig`
  - 2.3. Configure Amazon SageMaker Clarify as a processing job
  - 2.4. Run the Amazon SageMaker Clarify processing job
    - Exercise 2
  - 2.5. Run and review the Amazon SageMaker Clarify processing job on the unbalanced dataset
  - 2.6. Analyze unbalanced bias report
- 3. Balance the dataset by `product_category` and `sentiment`
- 4. Analyze bias on balanced dataset with Amazon SageMaker Clarify
  - 4.1. Configure a `DataConfig`
    - Exercise 3
  - 4.2. Configure `BiasConfig`
  - 4.3. Configure SageMaker Clarify as a processing job
  - 4.4. Run the Amazon SageMaker Clarify processing job
    - Exercise 4
  - 4.5. Run and review the Clarify processing job on the balanced dataset
  - 4.6. Analyze balanced bias report

First, let's install and import 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
import numpy as np
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c1/w2')

# 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
```

In [4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

1. Analyze the dataset

1.1. Create a pandas data frame from the CSV file

Create a pandas dataframe from each of the product categories and concatenate them into one.

In [5]:
```
!aws s3 cp 's3://dlai-practical-data-science/data/transformed/womens_clot
```

```
download: s3://dlai-practical-data-science/data/transformed/womens_clothi
ng_ecommerce_reviews_transformed.csv to ./womens_clothing_ecommerce_revie
ws_transformed.csv
```

In [6]:
```
path = './womens_clothing_ecommerce_reviews_transformed.csv'

df = pd.read_csv(path)
df.head()
```

Out[6]:

	sentiment	review_body	product_category
0	1	If this product was in petite i would get the...	Blouses
1	1	Love this dress! it's sooo pretty. i happene...	Dresses
2	0	I had such high hopes for this dress and reall...	Dresses
3	1	I love love love this jumpsuit. it's fun fl...	Pants
4	1	This shirt is very flattering to all due to th...	Blouses

As you saw in the previous lab, there are way more positive reviews than negative or neutral. Such a dataset is called unbalanced.

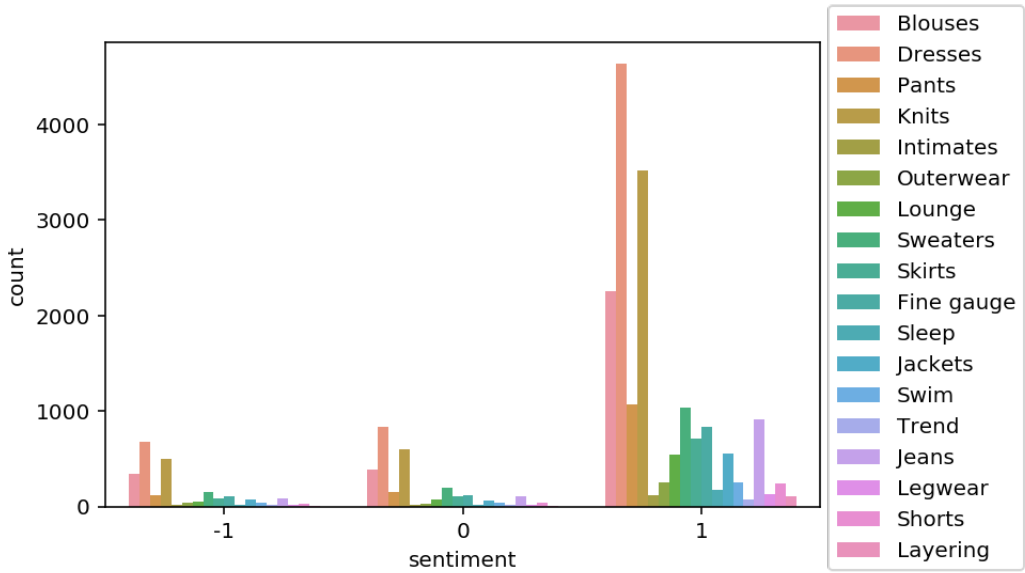
In this case, using a relatively small data subset you could visualize the occurring unbalances. At scale, you would need to perform bias analysis. Let's use this dataset as an example.

In [7]:

```
import seaborn as sns

sns.countplot(data=df, x='sentiment', hue='product_category')

plt.legend(loc='upper right',bbox_to_anchor=(1.3, 1.1))
```

Out[7]:

```
<matplotlib.legend.Legend at 0x7fc9b93cfc50>
```



1.2. Upload the dataset to S3 bucket

Upload the dataset to a private S3 bucket in a folder called `bias/unbalanced`.

In [8]:

```
data_s3_uri_unbalanced = sess.upload_data(bucket=bucket,
                                          key_prefix='bias/unbalanced',
                                          path='./womens_clothing_ecommerce_reviews_
data_s3_uri_unbalanced
```

Out[8]:

```
's3://sagemaker-us-east-1-449615851382/bias/unbalanced/womens_clothing_ec
ommerce_reviews_transformed.csv'
```

You can review the uploaded CSV file in the S3 bucket.

Instructions:

- open the link
- click on the S3 bucket name `sagemaker-us-east-1-ACCOUNT`
- go to the folder `bias/unbalanced`
- check the existence of the file `womens_clothing_ecommerce_reviews_transformed.csv`

In [9]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="top" href="https://s3.console.aws.amaz
```

Review Amazon S3 bucket

2. Analyze class imbalance on the dataset with Amazon SageMaker Clarify

Let's analyze bias in `sentiment` with respect to the `product_category` facet on the dataset.

2.1. Configure a `DataConfig`

Information about the input data needs to be provided to the processor. This can be done with the `DataConfig` of the Clarify container. It stores information about the dataset to be analyzed, for example the dataset file, its format, headers and labels.

Exercise 1

Configure a `DataConfig` for Clarify.

Instructions: Use `DataConfig` to configure the target column (`'sentiment'` label), data input (`data_s3_uri_unbalanced`) and output paths (`bias_report_unbalanced_output_path`) with their formats (header names and the dataset type):

```
data_config_unbalanced = clarify.DataConfig(
    s3_data_input_path=..., # S3 object path containing the
unbalanced dataset
    s3_output_path=..., # path to store the output
    label='...', # target column
    headers=df_unbalanced.columns.to_list(),
    dataset_type='text/csv'
)
```

In [10]:

```
from sagemaker import clarify

bias_report_unbalanced_output_path = 's3://{}/bias/generated_bias_report/

data_config_unbalanced = clarify.DataConfig(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    s3_data_input_path=data_s3_uri_unbalanced, # Replace None
    s3_output_path=bias_report_unbalanced_output_path, # Replace None
    label='sentiment', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    headers=df.columns.to_list(),
    dataset_type='text/csv'
)
```

2.2. Configure `BiasConfig`

Bias is measured by calculating a metric and comparing it across groups. To compute it, you will specify the required information in the `BiasConfig` API. SageMaker Clarify needs the sensitive columns (`facet_name`) and the desirable outcomes (`label_values_or_threshold`). Here `product_category` is the sensitive facet and the desired outcome is with the `sentiment==1` .

SageMaker Clarify can handle both categorical and continuous data for `label_values_or_threshold` . In this case you are using categorical data.

In [11]:

```
bias_config_unbalanced = clarify.BiasConfig(
    label_values_or_threshold=[1], # desired sentiment
    facet_name='product_category' # sensitive column (facet)
)
```

2.3. Configure Amazon SageMaker Clarify as a processing job

Now you need to construct an object called `SageMakerClarifyProcessor`. This allows you to scale the process of data bias detection using two parameters, `instance_count` and `instance_type`. `Instance_count` represents how many nodes you want in the distributor cluster during the data detection. `Instance_type` specifies the processing capability (compute capacity, memory capacity) available for each one of those nodes. 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 case outside of this lab.

In [12]:
```
clarify_processor_unbalanced = clarify.SageMakerClarifyProcessor(role=rol
                                                                 instance_count=1,
                                                                 instance_type='ml.m
                                                                 sagemaker_session=s
```

2.4. Run the Amazon SageMaker Clarify processing job

Exercise 2

Run the configured processing job to compute the requested bias `methods` of the input data

Instructions: Apply the `run_pre_training_bias` method to the configured Clarify processor, passing the configured input/output data (`data_config_unbalanced`), configuration of sensitive groups (`bias_config_unbalanced`) with the other job setup parameters:

```
clarify_processor_unbalanced.run_pre_training_bias(
    data_config=..., # configured input/output data
    data_bias_config=..., # configured sensitive groups
    methods=["CI", "DPL", "KL", "JS", "LP", "TVD", "KS"], #
selector of a subset of potential metrics
    wait=False, # whether the call should wait until the job
completes (default: True)
    logs=False # whether to show the logs produced by the job.
Only meaningful when wait is True (default: True)
)
```

In [13]:
```
clarify_processor_unbalanced.run_pre_training_bias(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    data_config=data_config_unbalanced, # Replace None
    data_bias_config=bias_config_unbalanced, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    methods=["CI", "DPL", "KL", "JS", "LP", "TVD", "KS"],
    wait=False,
    logs=False
)
```

```
Job Name:  Clarify-Pretraining-Bias-2022-08-22-12-51-10-265
Inputs:  [{'InputName': 'dataset', 'AppManaged': False, 'S3Input': {'S3Ur
i': 's3://sagemaker-us-east-1-449615851382/bias/unbalanced/womens_clothin
g_ecommerce_reviews_transformed.csv', 'LocalPath': '/opt/ml/processing/in
put/data', 'S3DataType': 'S3Prefix', 'S3InputMode': 'File', 'S3DataDistri
butionType': 'FullyReplicated', 'S3CompressionType': 'None'}}, {'InputNam
e': 'analysis_config', 'AppManaged': False, 'S3Input': {'S3Uri': 's3://sa
gemaker-us-east-1-449615851382/bias/generated_bias_report/unbalanced/anal
ysis_config.json', 'LocalPath': '/opt/ml/processing/input/config', 'S3Dat
aType': 'S3Prefix', 'S3InputMode': 'File', 'S3DataDistributionType': 'Ful
lyReplicated', 'S3CompressionType': 'None'}}]
Outputs:  [{'OutputName': 'analysis_result', 'AppManaged': False, 'S3Outp
ut': {'S3Uri': 's3://sagemaker-us-east-1-449615851382/bias/generated_bias
_report/unbalanced', 'LocalPath': '/opt/ml/processing/output', 'S3UploadM
ode': 'EndOfJob'}}]
```

In [14]:
```
run_unbalanced_bias_processing_job_name = clarify_processor_unbalanced.la
print(run_unbalanced_bias_processing_job_name)
```

```
Clarify-Pretraining-Bias-2022-08-22-12-51-10-265
```

2.5. Run and review the Amazon SageMaker Clarify processing job on the unbalanced dataset

Review the created Amazon SageMaker Clarify processing job and the Cloud Watch logs.

Instructions:

- open the link
- note that you are in the section Amazon SageMaker -> Processing jobs
- check the processing job name
- note which other properties of the processing job you can see in the console

In [15]:
```
from IPython.core.display import display, HTML

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

Review processing job

Instructions:

- open the link
- open the log stream with the name, which starts from the processing job name
- have a quick look at the log messages

In [16]:
```
from IPython.core.display import display, HTML

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

Review CloudWatch logs after about 5 minutes

In [17]:
```
running_processor = sagemaker.processing.ProcessingJob.from_processing_na
```

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

In [18]:
```
%%time

running_processor.wait(logs=False)
```

```
.....................................................................!CPU time
s: user 244 ms, sys: 53.7 ms, total: 298 ms
Wall time: 5min 27s
```

2.6. Analyze unbalanced bias report

In this run, you analyzed bias for `sentiment` relative to the `product_category` for the unbalanced data. Let's have a look at the bias report.

List the files in the output path `bias_report_unbalanced_output_path`:

In [19]:
```
!aws s3 ls $bias_report_unbalanced_output_path/
```

```
2022-08-22 12:57:24      31732 analysis.json
2022-08-22 12:51:11        346 analysis_config.json
2022-08-22 12:57:24     390923 report.html
2022-08-22 12:57:24     131783 report.ipynb
2022-08-22 12:57:24     150178 report.pdf
```

Download generated bias report from S3 bucket:

In [20]:
```
!aws s3 cp --recursive $bias_report_unbalanced_output_path ./generated_bi
```

```
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/unbalanced/analysis_config.json to generated_bias_report/unbalanced/ana
lysis_config.json
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/unbalanced/analysis.json to generated_bias_report/unbalanced/analysis.j
son
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/unbalanced/report.html to generated_bias_report/unbalanced/report.html
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/unbalanced/report.pdf to generated_bias_report/unbalanced/report.pdf
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/unbalanced/report.ipynb to generated_bias_report/unbalanced/report.ipyn
b
```

Review the downloaded bias report (in HTML format):

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

display(HTML('<b>Review <a target="blank" href="./generated_bias_report/u
```

Review unbalanced bias report

The bias report shows a number of metrics, but here you can focus on just two of them:

- Class Imbalance (CI). Measures the imbalance in the number of members between different facet values. Answers the question, does a `product_category` have disproportionately more reviews than others? Values of CI will become equal for even distribution between facets. Here, different CI values show the existence of imbalance.
- Difference in Positive Proportions in Labels (DPL). Measures the imbalance of positive outcomes between different facet values. Answers the question, does a `product_category` have disproportionately higher ratings than others? With the range over the interval from -1 to 1, if there is no bias, you want to see this value as close as possible to zero. Here, non-zero values indicate the imbalances.

3. Balance the dataset by `product_category` and `sentiment`

Let's balance the dataset by `product_category` and `sentiment`. Then you can configure and run SageMaker Clarify processing job to analyze the bias of it. Which metrics values do you expect to see in the bias report?

In [22]:
```
df_grouped_by = df.groupby(['product_category', 'sentiment'])
df_balanced = df_grouped_by.apply(lambda x: x.sample(df_grouped_by.size()
```

In [23]:
```
df_balanced
```

Out[23]:

product_category	sentiment		sentiment	review_body	product_category
Blouses	**-1**	**0**	-1	I loved the star pattern of this top but the c...	Blouses
		1	-1	The first time i wore it felt wonderful. had ...	Blouses
		2	-1	The print is very pretty but the fabric is sti...	Blouses
		3	-1	The shape is awkward too loose not flatterin...	Blouses
		4	-1	Cute top but started disintegrating after two ...	Blouses
...	**...**	**...**	...	...	...
Trend	**1**	**4**	1	Medium is equivalent to eu size 40. the pants ...	Trend
		5	1	I love his cardigan! i knew the small would be...	Trend
		6	1	I love the style and look oft this blouse but ...	Trend
		7	1	This easel caftan is simply amazing! the silho...	Trend
		8	1	I was on the fence about ordering this jacket ...	Trend

486 rows × 3 columns

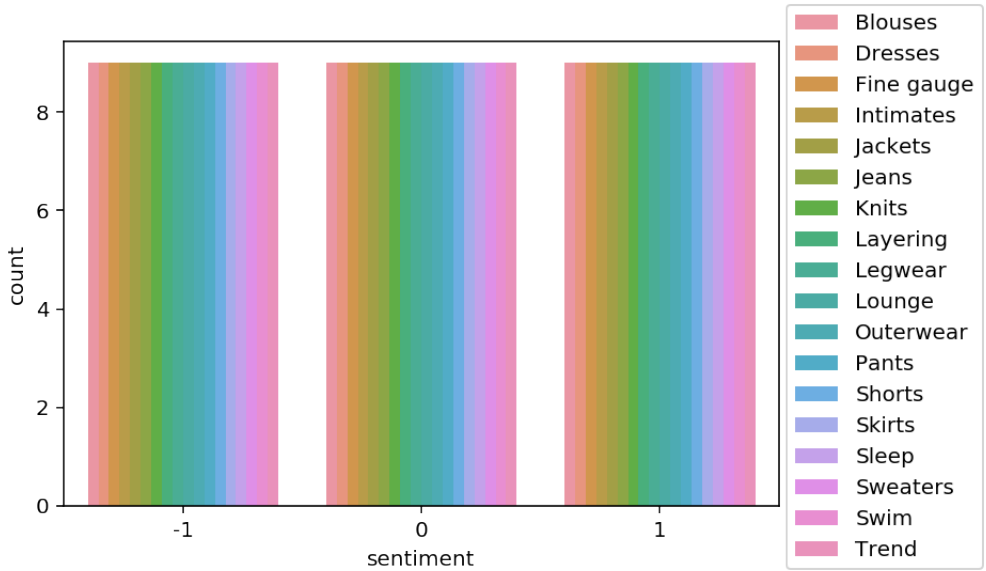
Visualize the distribution of review sentiment in the balanced dataset.

In [24]:

```
import seaborn as sns

sns.countplot(data=df_balanced, x='sentiment', hue='product_category')

plt.legend(loc='upper right',bbox_to_anchor=(1.3, 1.1))
```

Out[24]:

```
<matplotlib.legend.Legend at 0x7fc9b279e850>
```



4. Analyze bias on balanced dataset with Amazon SageMaker Clarify

Let's analyze bias in `sentiment` with respect to the `product_category` facet on your balanced dataset.

Save and upload balanced data to S3 bucket.

In [25]:
```
path_balanced = './womens_clothing_ecommerce_reviews_balanced.csv'
df_balanced.to_csv(path_balanced, index=False, header=True)

data_s3_uri_balanced = sess.upload_data(bucket=bucket, key_prefix='bias/b
data_s3_uri_balanced
```

Out[25]:
```
's3://sagemaker-us-east-1-449615851382/bias/balanced/womens_clothing_ecom
merce_reviews_balanced.csv'
```

You can review the uploaded CSV file in the S3 bucket and prefix `bias/balanced`.

In [26]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="top" href="https://s3.console.aws.amaz
```

Review Amazon S3 bucket

4.1. Configure a `DataConfig`

Exercise 3

Configure a `DataConfig` for Clarify to analyze bias on the balanced dataset.

Instructions: Pass the S3 object path containing the balanced dataset, the path to store the output (`bias_report_balanced_output_path`) and the target column. You can use exercise 1 as an example.

In [27]:
```
from sagemaker import clarify

bias_report_balanced_output_path = 's3://{}/bias/generated_bias_report/ba

data_config_balanced = clarify.DataConfig(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    s3_data_input_path=data_s3_uri_balanced, # Replace None
    s3_output_path=bias_report_balanced_output_path, # Replace None
    label='sentiment', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    headers=df_balanced.columns.to_list(),
    dataset_type='text/csv'
)
```

4.2. Configure `BiasConfig`

`BiasConfig` for the balanced dataset will have the same settings as before.

In [28]:
```
bias_config_balanced = clarify.BiasConfig(
    label_values_or_threshold=[1], # desired sentiment
    facet_name='product_category' # sensitive column (facet)
)
```

4.3. Configure SageMaker Clarify as a processing job

`SageMakerClarifyProcessor` object will also have the same parameters.

In [29]:
```
clarify_processor_balanced = clarify.SageMakerClarifyProcessor(role=role,
                                                               instance_count=1,
                                                               instance_type='ml.m
                                                               sagemaker_session=s
```

4.4. Run the Amazon SageMaker Clarify processing job

Exercise 4

Run the configured processing job for the balanced dataset.

Instructions: Apply the `run_pre_training_bias` method to the configured Clarify processor, passing the input/output data, configuration of sensitive groups with the other job setup parameters. You can use exercise 2 as an example.

In [30]:
```
clarify_processor_balanced.run_pre_training_bias(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    data_config=data_config_balanced, # Replace None
    data_bias_config=bias_config_balanced, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    methods=["CI", "DPL", "KL", "JS", "LP", "TVD", "KS"],
    wait=False,
    logs=False
)
```

```
Job Name:  Clarify-Pretraining-Bias-2022-08-22-13-01-08-932
Inputs:  [{'InputName': 'dataset', 'AppManaged': False, 'S3Input': {'S3Ur
i': 's3://sagemaker-us-east-1-449615851382/bias/balanced/womens_clothing_
ecommerce_reviews_balanced.csv', 'LocalPath': '/opt/ml/processing/input/d
ata', 'S3DataType': 'S3Prefix', 'S3InputMode': 'File', 'S3DataDistributio
nType': 'FullyReplicated', 'S3CompressionType': 'None'}}, {'InputName': '
analysis_config', 'AppManaged': False, 'S3Input': {'S3Uri': 's3://sagemak
er-us-east-1-449615851382/bias/generated_bias_report/balanced/analysis_co
nfig.json', 'LocalPath': '/opt/ml/processing/input/config', 'S3DataType':
'S3Prefix', 'S3InputMode': 'File', 'S3DataDistributionType': 'FullyReplic
ated', 'S3CompressionType': 'None'}}]
Outputs:  [{'OutputName': 'analysis_result', 'AppManaged': False, 'S3Outp
ut': {'S3Uri': 's3://sagemaker-us-east-1-449615851382/bias/generated_bias
_report/balanced', 'LocalPath': '/opt/ml/processing/output', 'S3UploadMod
e': 'EndOfJob'}}]
```

In [31]:
```
run_balanced_bias_processing_job_name = clarify_processor_balanced.latest
print(run_balanced_bias_processing_job_name)
```

```
Clarify-Pretraining-Bias-2022-08-22-13-01-08-932
```

4.5. Run and review the Clarify processing job on the balanced dataset

Review the results of the run following the links:

In [32]:
```
from IPython.core.display import display, HTML

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

Review processing job

In [33]:
```
from IPython.core.display import display, HTML

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

Review CloudWatch logs after about 5 minutes

In [34]:
```
running_processor = sagemaker.processing.ProcessingJob.from_processing_na
```

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

In [35]:
```
%%time

running_processor.wait(logs=False)
```

```
.....................................................................
!CPU times: user 292 ms, sys: 47.8 ms, total: 339 ms
Wall time: 6min 12s
```

4.6. Analyze balanced bias report

List the files in the output path `bias_report_balanced_output_path`:

In [36]:
```
!aws s3 ls $bias_report_balanced_output_path/
```

```
2022-08-22 13:07:32      29889 analysis.json
2022-08-22 13:01:09        346 analysis_config.json
2022-08-22 13:07:32     398677 report.html
2022-08-22 13:07:32     139537 report.ipynb
2022-08-22 13:07:32     152258 report.pdf
```

Download generated bias report from S3 bucket:

In [37]:
```
!aws s3 cp --recursive $bias_report_balanced_output_path ./generated_bias
```

```
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/balanced/analysis_config.json to generated_bias_report/balanced/analysi
s_config.json
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/balanced/analysis.json to generated_bias_report/balanced/analysis.json
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/balanced/report.html to generated_bias_report/balanced/report.html
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/balanced/report.ipynb to generated_bias_report/balanced/report.ipynb
download: s3://sagemaker-us-east-1-449615851382/bias/generated_bias_repor
t/balanced/report.pdf to generated_bias_report/balanced/report.pdf
```

Review the downloaded bias report (in HTML format):

In [38]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="blank" href="./generated_bias_report/b
```

Review balanced bias report

In this run, you analyzed bias for `sentiment` relative to the `product_category` for the balanced data. Note that the Class Imbalance (CI) metric is equal across all product categories for the target label, `sentiment`. And Difference in Positive Proportions in Labels (DPL) metric values are zero.

Upload the notebook into S3 bucket for grading purposes.

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

In [39]:
```
!aws s3 cp ./C1_W2_Assignment.ipynb s3://$bucket/C1_W2_Assignment_Learner
```

```
upload: ./C1_W2_Assignment.ipynb to s3://sagemaker-us-east-1-449615851382
/C1_W2_Assignment_Learner.ipynb
```

Please go to the main lab window and click on `Submit` button (see the `Finish the lab` section of the instructions).

In [ ]:
```
```

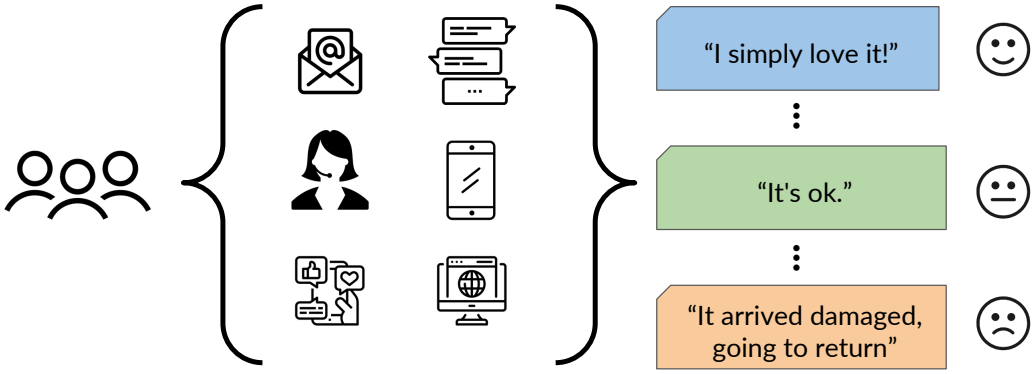
DeepLearning.AI

aws

Practical Data Science

Use AutoML to Train a Text Classifier

Multi-class classification for sentiment analysis of product reviews



Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Automated Machine Learning

(AutoML)

Model Building Challenges

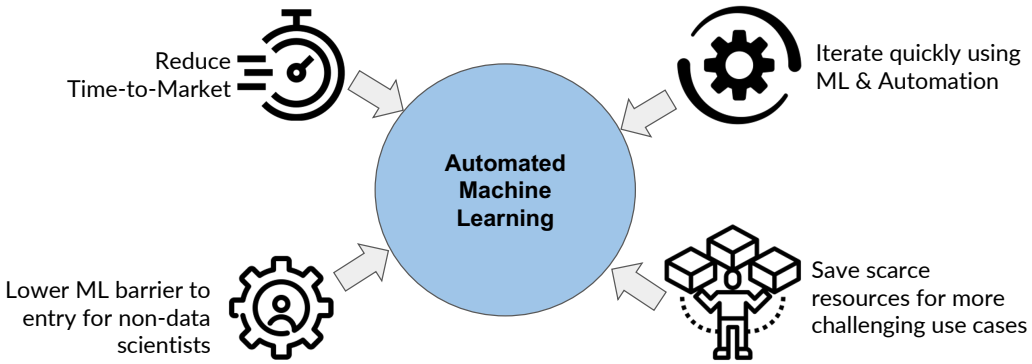
Ability to reduce Time-to-Market

Ability to iterate quickly

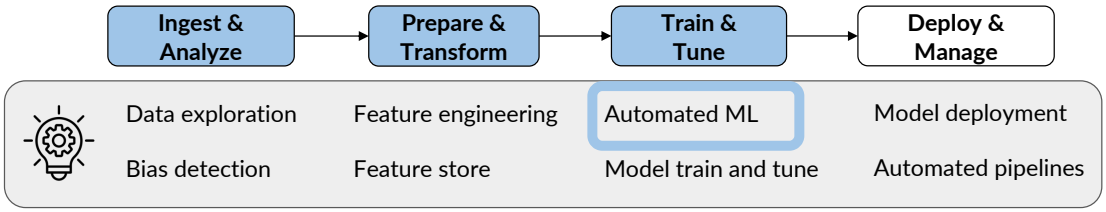
Lack of ML skillsets

Ability to optimize scarce resources & skillsets

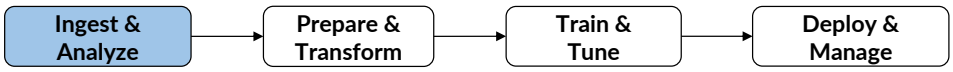
Automated Machine Learning



Machine Learning Workflow

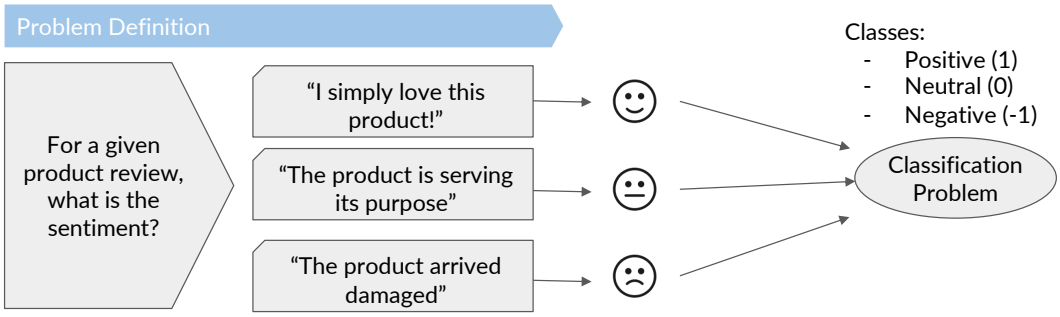


Data Preparation

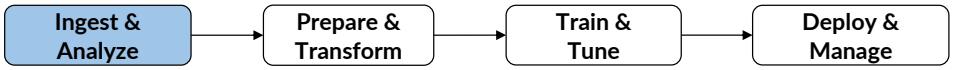


Data Analysis:

Collecting statistics, such as missing entries, quantiles, skewness, correlation with the target.

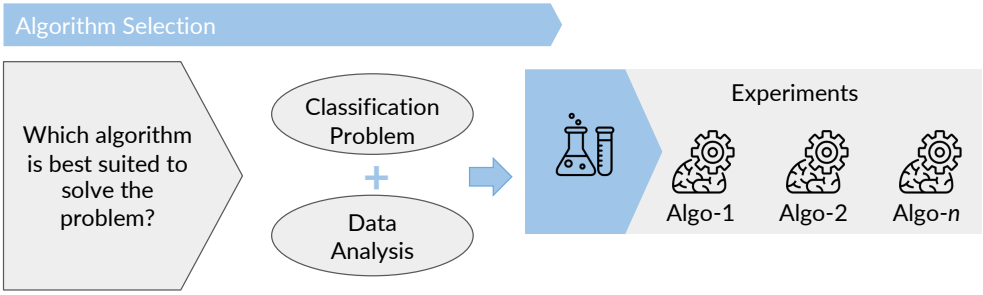


Data Preparation



Data Analysis:

Collecting statistics, such as missing entries, quantiles, skewness, correlation with the target.



Data Preparation

Ingest & Analyze → Prepare & Transform → Train & Tune → Deploy & Manage

Data Analysis:

Collecting statistics, such as missing values, quantiles, skewness, correlation with the target.

Dataset Schema Detection

Numeric	Categorical	Numeric
review_id	**review_text**	**sentiment**
001	"I simply love this product!"	1
002	"The product is serving its purpose"	0
003	"The Product arrived damaged"	-1

Data Preparation

Ingest & Analyze → Prepare & Transform → Train & Tune → Deploy & Manage

Data Transformation:

How should data be transformed so that the model can predict as accurately as possible?

review_id	review_text	sentiment
001	"I simply love this product!"	1
002	"The product is serving its purpose"	0
003	"The Product arrived damaged"	-1

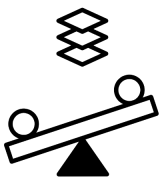
Too Many Unique Values
= Treat as Text

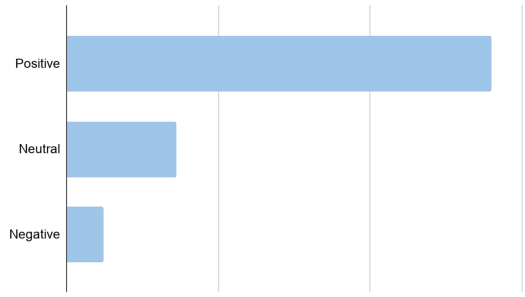
Data Preparation

Ingest & Analyze → Prepare & Transform → Train & Tune → Deploy & Manage

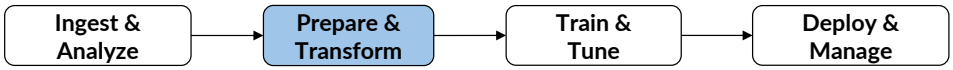
Class Imbalance:

How to identify and handle potential class imbalance?

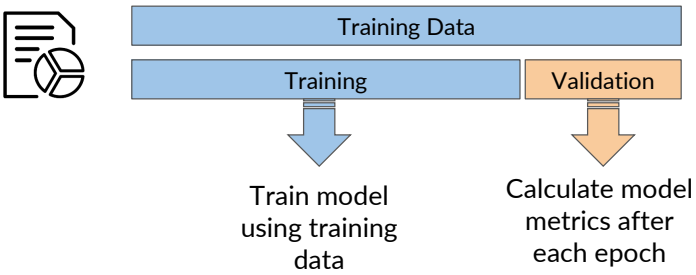




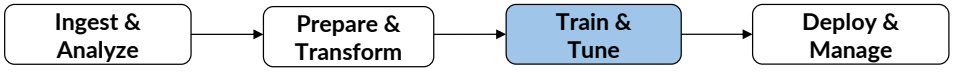
Data Preparation: Train and Validation Data Splits



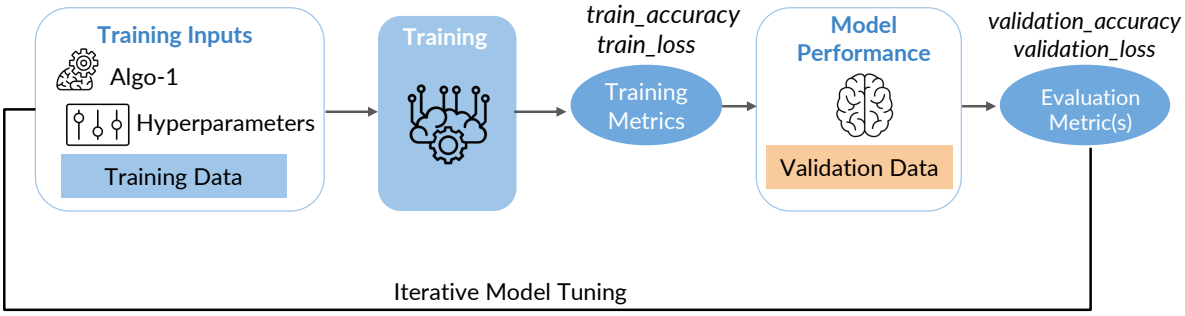
Train-Validation Splits:
Splitting prepared data for model training, model performance, and final model evaluation



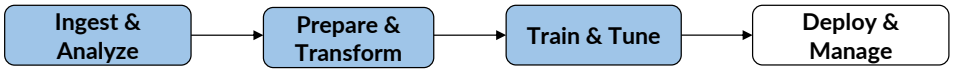
Model Training & Tuning



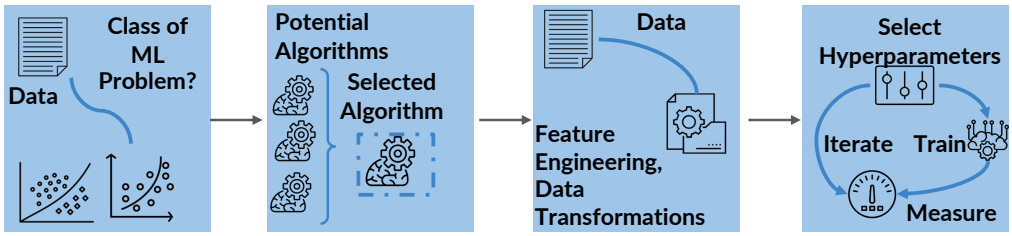
Model Training:
Fit the model to your data



AutoML



AutoML aims at automating the process of building a model



Problem Identification | Algorithm Selection | Data Preprocessing | Hyperparameter Tuning

Scenarios for AutoML

Build models without any ML expertise

- Empower more people in your organization: software developers, business people
- Let experts focus on **hard problems**

Experiment and build models at scale

- Thousands of data sets can be modeled without human intervention
- Let experts focus on **new problems**

Automate the majority of the work, then tweak

- Data cleaning, feature engineering, feature selection, etc.
- Let experts focus on high value tasks such as **domain knowledge**, and **error analysis**.

Transparency and Control are Important

Get the **best model** only

- Hard to understand it
- Hard to reproduce it manually

Get the **best model**, **all candidates**, **full source code**

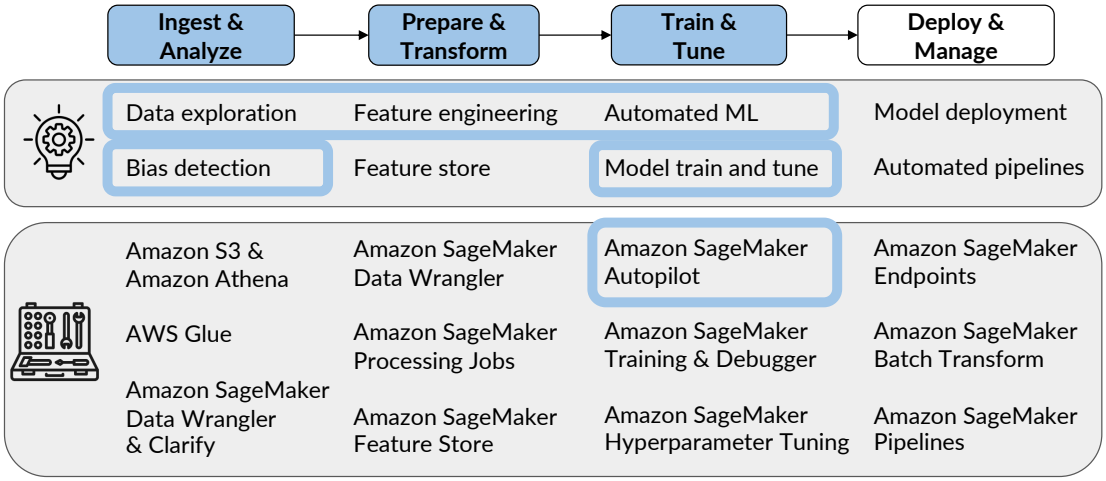
- Understand how the model was built
- Keep tweaking for extra performance

AutoML

with
Amazon SageMaker Autopilot

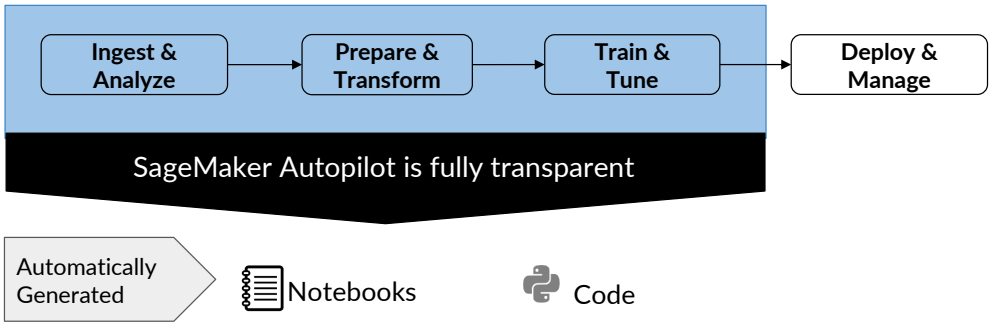
Introduction

Machine Learning Workflow

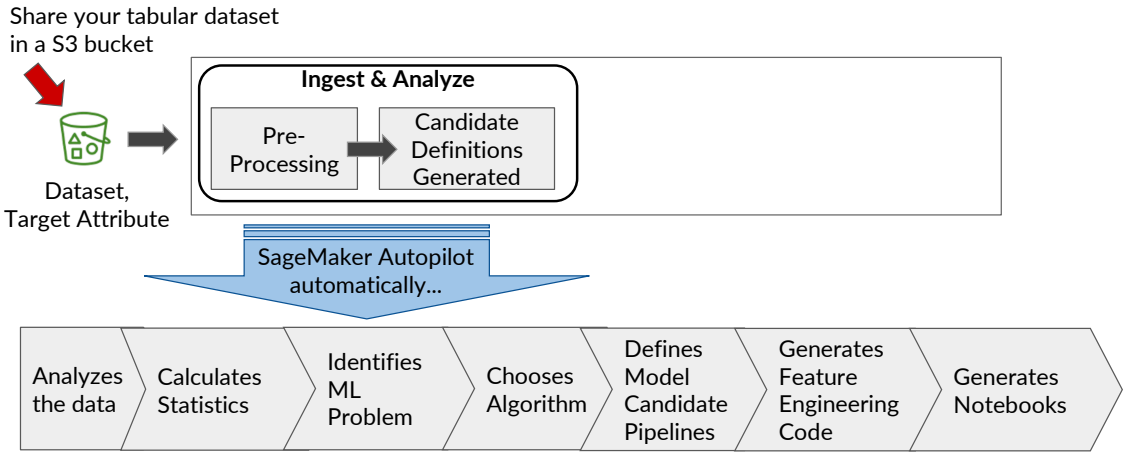


AutoML with Amazon SageMaker Autopilot

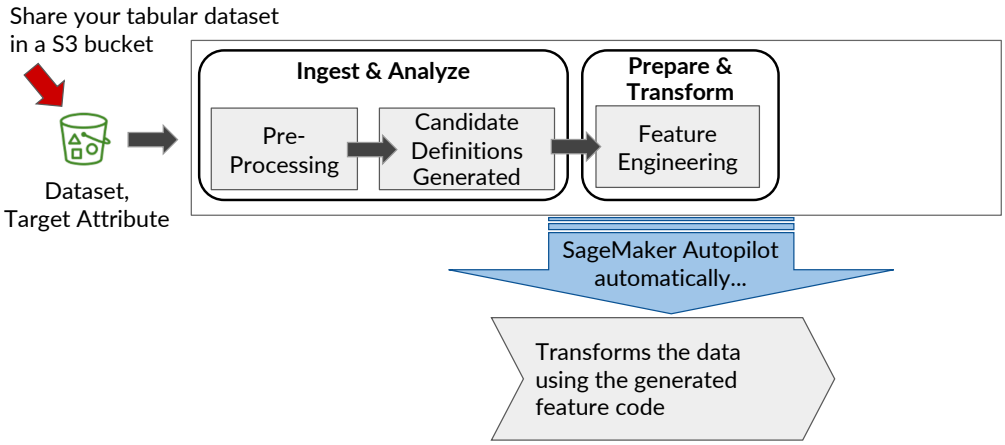
Amazon SageMaker Autopilot covers all steps:



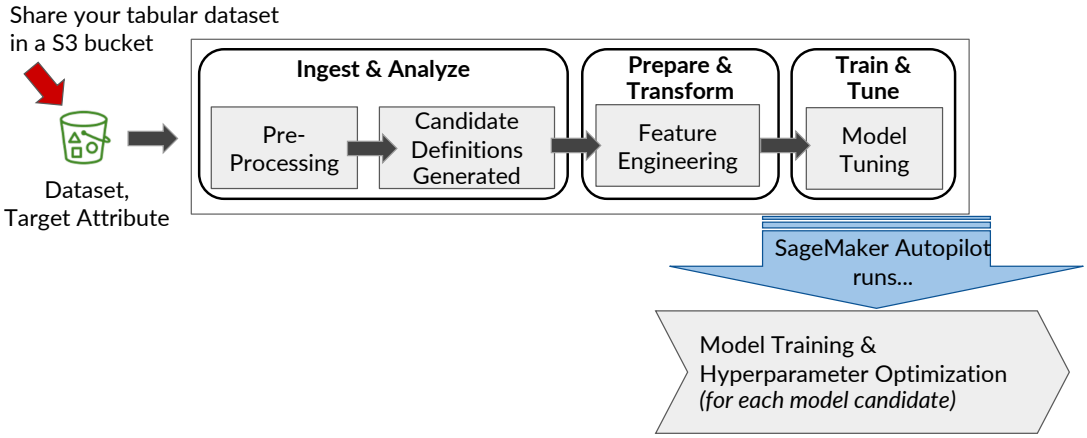
Amazon SageMaker Autopilot at a High-Level



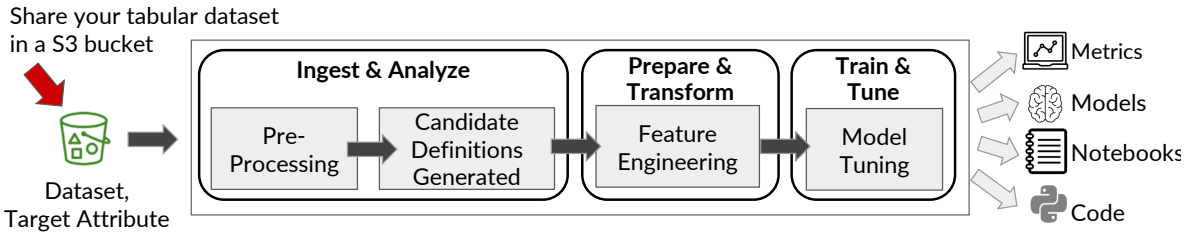
Amazon SageMaker Autopilot at a High-Level



Amazon SageMaker Autopilot at a High-Level



Amazon SageMaker Autopilot at a High-Level



SageMaker Autopilot shares...

- All metrics
- Leaderboard of model candidates
- Notebooks
- Code

AutoML

with
Amazon SageMaker Autopilot

Running Experiments

Amazon SageMaker Autopilot Notebook Overview

Use Case: Analyze Customer Sentiment

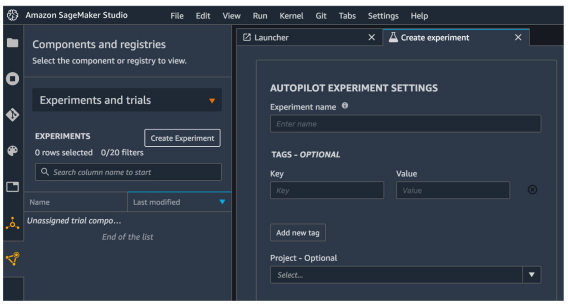
Goal: Use SageMaker Autopilot to find the optimal feature transformations, algorithm, and hyperparameters to produce a best performing model allowing us to predict our label (sentiment) based on product reviews (review_body)

sentiment	review_body
-1	This is bad.
0	This is OK.
1	This is great!

Interacting with Amazon SageMaker Autopilot

Programmatically:
1. AWS CLI
2. AWS SDK
3. Amazon SageMaker Python SDK

~OR~



Amazon SageMaker Studio

Launch the Amazon SageMaker Autopilot Job

```
automl = sagemaker.automl.automl.AutoML(
    target_attribute_name=...
    output_path=..,
    max_candidates=3,
    role=role,
    max_runtime_per_training_job_in_seconds=1200,
    total_job_runtime_in_seconds=7200 # max automl job runtime in seconds
)

automl.fit(
    inputs=...,
)
```

Attribute to predict

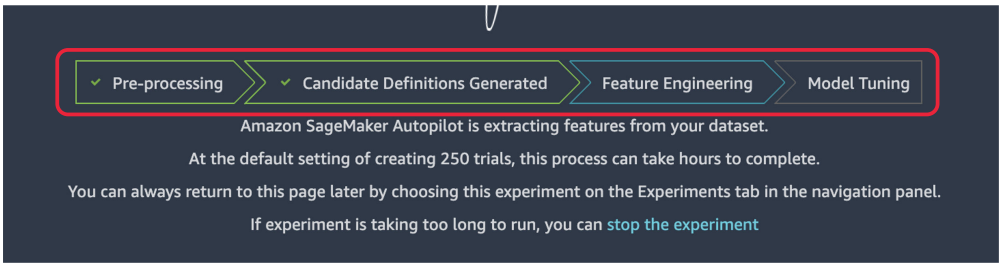
Job completion criteria

Max. training job run time

Max. AutoML job runtime

Specify input data

Monitor Progress in Amazon SageMaker Studio



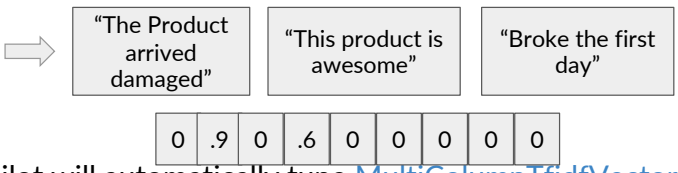
API → *DescribeAutoMLJob*

Generated Code for Feature Engineering

- SageMaker Autopilot automatically performs data exploration and prepares the data for the problem type

Example(s):

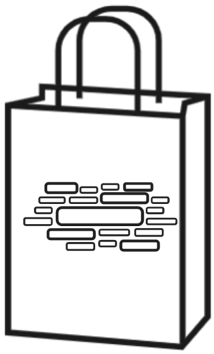
Transform text features using MultiColumnTfidfVectorizer



- SageMaker Autopilot will automatically tune MultiColumnTfidfVectorizer parameters

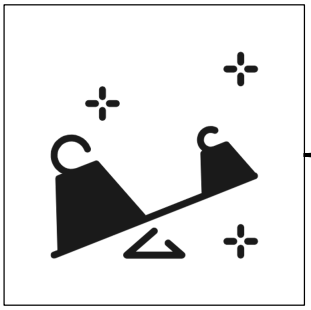
Bag-of-Words: Text as Vectors

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



Term	Term Count
it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
...	...

Text Mining: Measuring Word Importance



- Statistical measure of word importance in corpus
- ↑ proportionally to the number of times a word appears in document
- ↓ by the frequency of the word in the corpus
- tf-idf: term frequency-inverse document frequency

Computing Term Frequency (TF)

t	term
d	document
D	corpus

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

Computing Inverse Document Frequency (IDF)

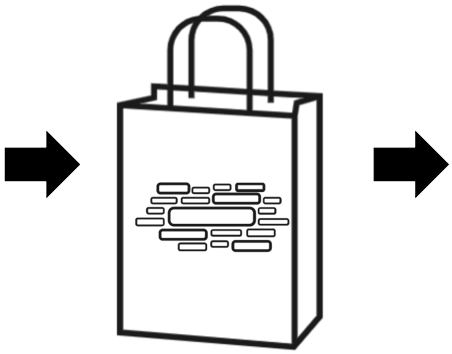
t	term
d	document
D	corpus

$$\text{idf}(t, D) = \log\left(\frac{|D|}{|\{d \in D : t \in d\}|}\right)$$

Putting It All Together: TF-IDF

$$\text{tf-idf}(t, d, D) = \text{tf}(t, d) \cdot \text{idf}(t, D)$$

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



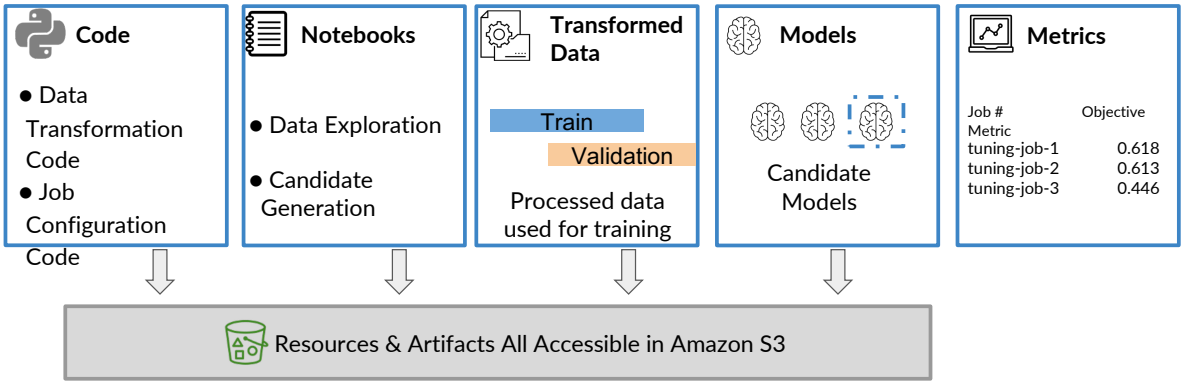
Term	TF / IDF
it	0.06
I	0.05
the	0.01
to	0.03
and	0.03
seen	0.04
yet	0.01
...	...

AutoML

with
Amazon SageMaker Autopilot

Evaluating Output

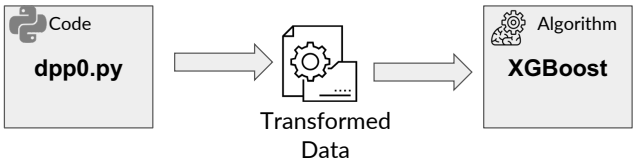
SageMaker Autopilot Generates Resources & Artifacts



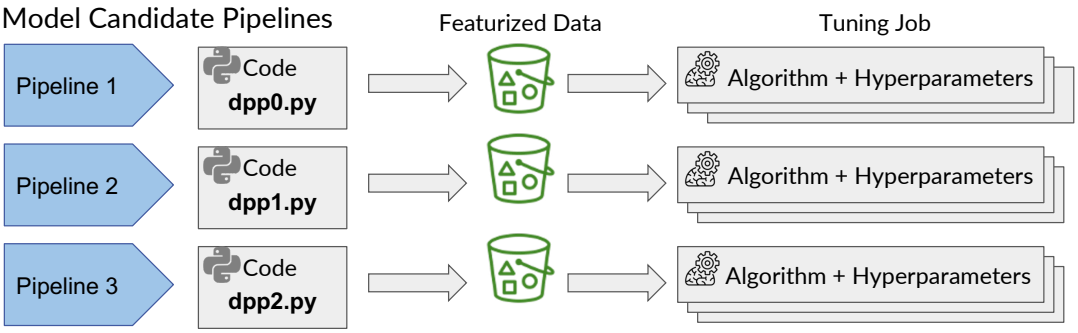
Model Candidate Pipelines

A model candidate pipeline is composed of

- the feature engineering code (i.e. dpp0.py)
- and an algorithm (i.e. XGBoost).



Executing Model Candidate Pipelines




```
automl_interactive_runner.fit_data_transformers(parallel_jobs=7)
```

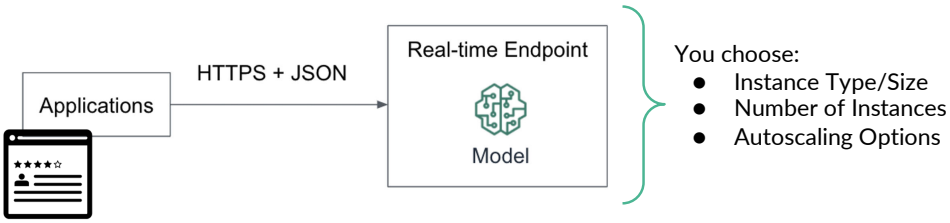
Model Hosting

Introduction

Host a Model Endpoint

Deploy the model to serve predictions in real-time.

- Optimized for **low latency** of model predictions
- Example: As product reviews are coming in through various online channels, you want to predict the sentiment



Deploy Inference Pipeline

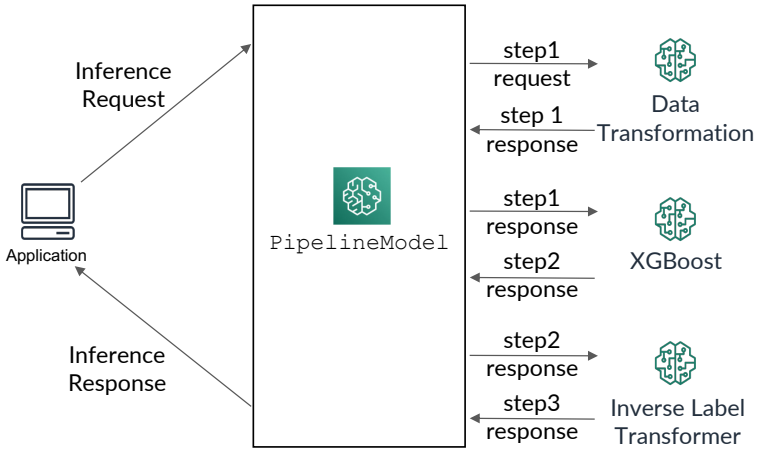
```
pipeline_model.deploy(initial_instance_count=1,
                      instance_type='ml.m5.2xlarge',
                      endpoint_name=pipeline_model.name,
                      wait=True)
```

Congratulations! Now you could visit the sagemaker endpoint console page to find the deployed endpoint (it'll take a few minutes to be in service).

The `PipelineModel` has multiple containers of the following:

- **Data Transformation Container:** a container built from the model we selected and trained during the data transformer sections
- **Algorithm Container:** a container built from the trained model we selected above from the best HPO training job.
- **Inverse Label Transformer Container:** a container that converts numerical intermediate prediction value back to non-numerical label value.

Inference Pipeline



Train a model with Amazon SageMaker Autopilot

Introduction

In this lab, you will use Amazon Sagemaker Autopilot to train a BERT-based natural language processing (NLP) model. The model will analyze customer feedback and classify the messages into positive (1), neutral (0) and negative (-1) sentiment.

Table of Contents

- 1. Review transformed dataset
- 2. Configure the Autopilot job
  - 2.1. Upload data to S3 bucket
  - 2.2. S3 output for generated assets
  - 2.3. Configure the Autopilot job
    - Exercise 1
- 3. Launch the Autopilot job
  - Exercise 2
- 4. Track Autopilot job progress
  - 4.1. Autopilot job description
  - 4.2. Autopilot job status
  - 4.3. Review the SageMaker processing jobs
  - 4.4. Wait for the data analysis step to finish
  - 4.5. View generated notebooks
    - Exercise 3
    - Exercise 4
- 5. Feature engineering
  - Exercise 5
- 6. Model training and tuning
  - 6.1. Wait for training and tuning
    - Exercise 6
  - 6.2. Compare model candidates
    - Exercise 7
  - 6.3. Review best candidate
    - Exercise 8
- 7. Review all output in S3 bucket
- 8. Deploy and test best candidate model
  - 8.1. Deploy best candidate model
  - 8.2. Test the model

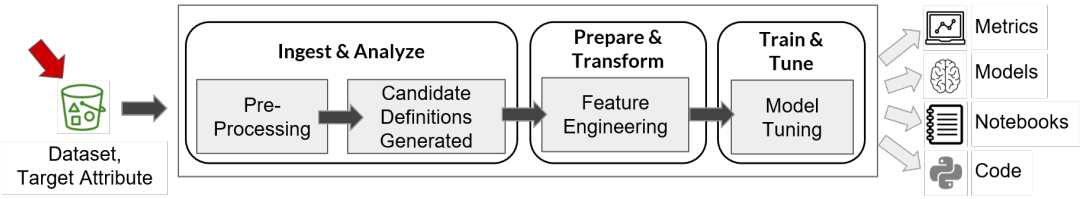
Amazon SageMaker Autopilot automatically trains and tunes the best machine

learning models for classification or regression, based on your data while allowing to maintain full control and visibility.

SageMaker Autopilot will inspect the raw dataset, apply feature processors, pick the best set of algorithms, train and tune multiple models, and then rank the models based on performance - all with just a few clicks. Autopilot transparently generates a set of Python scripts and notebooks for a complete end-to-end pipeline including data analysis, candidate generation, feature engineering, and model training/tuning.

SageMaker Autopilot job consists of the following high-level steps:

- *Data analysis* where the data is summarized and analyzed to determine which feature engineering techniques, hyper-parameters, and models to explore.
- *Feature engineering* where the data is scrubbed, balanced, combined, and split into train and validation.
- *Model training and tuning* where the top performing features, hyper-parameters, and models are selected and trained.



These re-usable scripts and notebooks give us full visibility into how the model candidates were created. Since Autopilot integrates natively with SageMaker Studio, we can visually explore the different models generated by SageMaker Autopilot.

SageMaker Autopilot can be used by people without machine learning experience to automatically train a model from a dataset. Additionally, experienced developers can use Autopilot to train a baseline model from which they can iterate and manually improve.

Autopilot is available through the SageMaker Studio UI and AWS Python SDK. In this notebook, you will use the AWS Python SDK to train a series of text-classification models and deploy the model with the highest accuracy.

For more details on Autopilot, have a look at this **Amazon Science Publication**.

Use case: analyze customer sentiment

Customer feedback appears across many channels including social media and partner websites. As a company, you want to capture this valuable product feedback to spot negative trends and improve the situation, if needed. Here you will train a model to classify the feedback messages into positive (1), neutral (0) and negative (-1) sentiment.

First, let's install and import 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
import numpy as np
import botocore
import time
import json

config = botocore.config.Config(user_agent_extra='dlai-pds/c1/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
```

In [4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

1. Review transformed dataset

Let's transform the dataset into a format that Autopilot recognizes. Specifically, a comma-separated file of `label,features` as shown here:

```
sentiment,review_body
-1,"this is bad"
0,"this is ok"
1,"this is great"
...
```

Sentiment is one of three classes: negative (-1), neutral (0), or positive (1). Autopilot requires that the target variable, `sentiment` is first and the set of features, just `review_body` in this case, come next.

In [5]:
```
!aws s3 cp 's3://dlai-practical-data-science/data/balanced/womens_clothin
```

```
download: s3://dlai-practical-data-science/data/balanced/womens_clothing_
ecommerce_reviews_balanced.csv to ./womens_clothing_ecommerce_reviews_bal
anced.csv
```

In [6]:
```
path = './womens_clothing_ecommerce_reviews_balanced.csv'

df = pd.read_csv(path, delimiter=',')
df.head()
```

Out[6]:

	sentiment	review_body	product_category
0	-1	This suit did nothing for me. the top has zero...	Swim
1	-1	Like other reviewers i saw this dress on the ...	Dresses
2	-1	I wish i had read the reviews before purchasin...	Knits
3	-1	I ordered these pants in my usual size (xl) an...	Legwear
4	-1	I noticed this top on one of the sales associa...	Knits

In [7]:
```
path_autopilot = './womens_clothing_ecommerce_reviews_balanced_for_autopi

df[['sentiment', 'review_body']].to_csv(path_autopilot,
                                        sep=',',
                                        index=False)
```

2. Configure the Autopilot job

2.1. Upload data to S3 bucket

In [8]:
```
autopilot_train_s3_uri = sess.upload_data(bucket=bucket, key_prefix='auto
autopilot_train_s3_uri
```

Out[8]:
```
's3://sagemaker-us-east-1-917875584351/autopilot/data/womens_clothing_eco
mmerce_reviews_balanced_for_autopilot.csv'
```

Check the existence of the dataset in this S3 bucket folder:

In [9]:
```
!aws s3 ls $autopilot_train_s3_uri
```

```
2022-08-23 13:49:40    2253749 womens_clothing_ecommerce_reviews_balanced
_for_autopilot.csv
```

2.2. S3 output for generated assets

Set the S3 output path for the Autopilot outputs. This includes Jupyter notebooks (analysis), Python scripts (feature engineering), and trained models.

In [10]:
```
model_output_s3_uri = 's3://{}/autopilot'.format(bucket)

print(model_output_s3_uri)
```

```
s3://sagemaker-us-east-1-917875584351/autopilot
```

2.3. Configure the Autopilot job

Create the Autopilot job name.

In [11]:
```
import time

timestamp = int(time.time())

auto_ml_job_name = 'automl-dm-{}'.format(timestamp)
```

When configuring our Autopilot job, you need to specify the maximum number of candidates, `max_candidates`, to explore as well as the input/output S3 locations and target column to predict. In this case, you want to predict `sentiment` from the review text.

Exercise 1

Configure the Autopilot job.

Instructions: Create an instance of the `sagemaker.automl.automl.AutoML` estimator class passing the required configuration parameters. Target attribute for predictions here is `sentiment`.

```
automl = sagemaker.automl.automl.AutoML(
    target_attribute_name='...', # the name of the target
attribute for predictions
    base_job_name=..., # Autopilot job name
    output_path=..., # output data path
    max_candidates=..., # maximum number of candidates
    sagemaker_session=sess,
    role=role,
    max_runtime_per_training_job_in_seconds=1200,
    total_job_runtime_in_seconds=7200
)
```

In [14]:
```
max_candidates = 3

automl = sagemaker.automl.automl.AutoML(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    target_attribute_name='sentiment', # Replace None
    base_job_name=auto_ml_job_name, # Replace None
    output_path=model_output_s3_uri, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    max_candidates=max_candidates,
    sagemaker_session=sess,
    role=role,
    max_runtime_per_training_job_in_seconds=1200,
    total_job_runtime_in_seconds=7200
)
```

3. Launch the Autopilot job

Exercise 2

Launch the Autopilot job.

Instructions: Call `fit` function of the configured estimator passing the S3 bucket input data path and the Autopilot job name.

```
automl.fit(
    ..., # input data path
    job_name=auto_ml_job_name, # Autopilot job name
    wait=False,
    logs=False
)
```

In [15]:
```
automl.fit(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    autopilot_train_s3_uri, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    job_name=auto_ml_job_name,
    wait=False,
    logs=False
)
```

4. Track Autopilot job progress

Once the Autopilot job has been launched, you can track the job progress directly from the notebook using the SDK capabilities.

4.1. Autopilot job description

Function `describe_auto_ml_job` of the Amazon SageMaker service returns the information about the AutoML job in dictionary format. You can review the response syntax and response elements in the **documentation**.

In [16]:
```
job_description_response = automl.describe_auto_ml_job(job_name=auto_ml_j
```

4.2. Autopilot job status

To track the job progress you can use two response elements: `AutoMLJobStatus` and `AutoMLJobSecondaryStatus`, which correspond to the primary (Completed | InProgress | Failed | Stopped | Stopping) and secondary (AnalyzingData | FeatureEngineering | ModelTuning etc.) job states respectively. To see if the AutoML job has started, you can check the existence of the `AutoMLJobStatus` and `AutoMLJobSecondaryStatus` elements in the job description response.

In this notebook, you will use the following scheme to track the job progress:

```
# check if the job is still at certain stage
while [check 'AutoMLJobStatus' and 'AutoMLJobSecondaryStatus']
in job_description_response:
    # update the job description response
    job_description_response =
automl.describe_auto_ml_job(AutoMLJobName=auto_ml_job_name)
    # print the message the Autopilot job is in the stage ...
    print([message])
    # git a time step to check the status again
    sleep(15)
print("Autopilot job complete...")
```

In [17]:
```
while 'AutoMLJobStatus' not in job_description_response.keys() and 'AutoM
    job_description_response = automl.describe_auto_ml_job(job_name=auto_
    print('[INFO] Autopilot job has not yet started. Please wait. ')
    # function `json.dumps` encodes JSON string for printing.
    print(json.dumps(job_description_response, indent=4, sort_keys=True,
    print('[INFO] Waiting for Autopilot job to start...')
    sleep(15)

print('[OK] AutoML job started.')
```

```
[OK] AutoML job started.
```

4.3. Review the SageMaker processing jobs

The Autopilot creates required SageMaker processing jobs during the run:

- First processing job (data splitter) checks the data sanity, performs stratified shuffling and splits the data into training and validation.
- Second processing job (candidate generator) first streams through the data to compute statistics for the dataset. Then, uses these statistics to identify the problem type, and possible types of every column-predictor: numeric, categorical, natural language, etc.

In [18]:
```
from IPython.core.display import display, HTML

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

Review processing jobs

You can review the updates on that page during the run of the Autopilot job.

4.4. Wait for the data analysis step to finish

Here you will use the same scheme as above to check the completion of the data analysis step. This step can be identified with the (primary) job status value `InProgress` and secondary job status values `Starting` and then `AnalyzingData`.

This cell will take approximately 10 minutes to run.

In [19]:
```
%%time

job_status = job_description_response['AutoMLJobStatus']
job_sec_status = job_description_response['AutoMLJobSecondaryStatus']

if job_status not in ('Stopped', 'Failed'):
    while job_status in ('InProgress') and job_sec_status in ('Starting',
        job_description_response = automl.describe_auto_ml_job(job_name=a
        job_status = job_description_response['AutoMLJobStatus']
        job_sec_status = job_description_response['AutoMLJobSecondaryStat
        print(job_status, job_sec_status)
        time.sleep(15)
    print('[OK] Data analysis phase completed.\n')

print(json.dumps(job_description_response, indent=4, sort_keys=True, defa
```

```
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
InProgress AnalyzingData
```

```
InProgress AnalyzingData
InProgress FeatureEngineering
[OK] Data analysis phase completed.

{
    "AutoMLJobArn": "arn:aws:sagemaker:us-east-1:917875584351:automl-job/
automl-dm-1661262662",
    "AutoMLJobArtifacts": {
        "CandidateDefinitionNotebookLocation": "s3://sagemaker-us-east-1-
917875584351/autopilot/automl-dm-1661262662/sagemaker-automl-candidates/a
utoml-dm-1661262662-pr-1-61719c565070432bb34d947dd84accb25aafb/notebooks/
SageMakerAutopilotCandidateDefinitionNotebook.ipynb",
        "DataExplorationNotebookLocation": "s3://sagemaker-us-east-1-9178
75584351/autopilot/automl-dm-1661262662/sagemaker-automl-candidates/autom
l-dm-1661262662-pr-1-61719c565070432bb34d947dd84accb25aafb/notebooks/Sage
MakerAutopilotDataExplorationNotebook.ipynb"
    },
    "AutoMLJobConfig": {
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": 7200,
            "MaxCandidates": 3,
            "MaxRuntimePerTrainingJobInSeconds": 1200
        },
        "SecurityConfig": {
            "EnableInterContainerTrafficEncryption": false
        }
    },
    "AutoMLJobName": "automl-dm-1661262662",
    "AutoMLJobSecondaryStatus": "FeatureEngineering",
    "AutoMLJobStatus": "InProgress",
    "CreationTime": "2022-08-23 13:58:13.486000+00:00",
    "GenerateCandidateDefinitionsOnly": false,
    "InputDataConfig": [
        {
            "ChannelType": "training",
            "ContentType": "text/csv;header=present",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://sagemaker-us-east-1-917875584351/autop
ilot/data/womens_clothing_ecommerce_reviews_balanced_for_autopilot.csv"
                }
            },
            "TargetAttributeName": "sentiment"
        }
    ],
    "LastModifiedTime": "2022-08-23 14:07:30.547000+00:00",
    "OutputDataConfig": {
        "S3OutputPath": "s3://sagemaker-us-east-1-917875584351/autopilot"
    },
    "ResolvedAttributes": {
        "AutoMLJobObjective": {
            "MetricName": "Accuracy"
        },
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": 7200,
            "MaxCandidates": 3,
            "MaxRuntimePerTrainingJobInSeconds": 1200
        },
        "ProblemType": "MulticlassClassification"
    },
```

```
    "ResponseMetadata": {
        "HTTPHeaders": {
            "content-length": "1811",
            "content-type": "application/x-amz-json-1.1",
            "date": "Tue, 23 Aug 2022 14:07:35 GMT",
            "x-amzn-requestid": "9c31db68-c5dc-4fcf-84e1-3e59debdd9da"
        },
        "HTTPStatusCode": 200,
        "RequestId": "9c31db68-c5dc-4fcf-84e1-3e59debdd9da",
        "RetryAttempts": 0
    },
    "RoleArn": "arn:aws:iam::917875584351:role/sagemaker-studio-vpc-firew
all-us-east-1-sagemaker-execution-role"
}
CPU times: user 357 ms, sys: 44.9 ms, total: 402 ms
Wall time: 7min 20s
```

Wait for Autopilot to finish generating the notebooks.

4.5. View generated notebooks

Once data analysis is complete, SageMaker AutoPilot generates two notebooks:

- Data exploration
- Candidate definition

Notebooks are included in the AutoML job artifacts generated during the run. Before checking the existence of the notebooks, you can check if the artifacts have been generated.

Exercise 3

Check if the Autopilot job artifacts have been generated.

Instructions: Use status check scheme described above. The generation of artifacts can be identified by existence of `AutoMLJobArtifacts` element in the keys of the job description response.

In [21]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
# get the information about the running Autopilot job
job_description_response = automl.describe_auto_ml_job(job_name=auto_ml_j

# keep in the while loop until the Autopilot job artifacts will be genera
while 'AutoMLJobArtifacts' not in job_description_response.keys(): # Repl
    # update the information about the running Autopilot job
    job_description_response = automl.describe_auto_ml_job(job_name=auto_
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    print('[INFO] Autopilot job has not yet generated the artifacts. Plea
    print(json.dumps(job_description_response, indent=4, sort_keys=True,
    print('[INFO] Waiting for AutoMLJobArtifacts...')
    time.sleep(15)

print('[OK] AutoMLJobArtifacts generated.')
```

```
[OK] AutoMLJobArtifacts generated.
```

Wait for Autopilot to make the notebooks available.

Exercise 4

Check if the notebooks have been created.

Instructions: Use status check scheme described above. Notebooks creation can be identified by existence of `DataExplorationNotebookLocation` element in the keys of the `job_description_response['AutoMLJobArtifacts']` dictionary.

In [22]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
# get the information about the running Autopilot job
job_description_response = automl.describe_auto_ml_job(job_name=auto_ml_j

# keep in the while loop until the notebooks will be created
while 'DataExplorationNotebookLocation' not in job_description_response['
    # update the information about the running Autopilot job
    job_description_response = automl.describe_auto_ml_job(job_name=auto_
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    print('[INFO] Autopilot job has not yet generated the notebooks. Plea
    print(json.dumps(job_description_response, indent=4, sort_keys=True,
    print('[INFO] Waiting for DataExplorationNotebookLocation...')
    time.sleep(15)

print('[OK] DataExplorationNotebookLocation found.')
```

```
[OK] DataExplorationNotebookLocation found.
```

Review the generated resources in S3 directly. Following the link, you can find the notebooks in the folder `notebooks` and download them by clicking on object `Actions` / `Object actions` -> `Download as` / `Download`.

In [23]:
```
from IPython.core.display import display, HTML

generated_resources = job_description_response['AutoMLJobArtifacts']['Dat
download_path = generated_resources.rsplit('/notebooks/SageMakerAutopilot
job_id = download_path.rsplit('/', 1)[-1]

if not job_id:
    print('No AutoMLJobArtifacts found.')
else:
    display(HTML('<b>Review <a target="blank" href="https://s3.console.aw
```

Review generated notebooks in S3 bucket

5. Feature engineering

Exercise 5

Check the completion of the feature engineering step.

Instructions: Use status check scheme described above. Feature engineering step can be identified with the (primary) job status value `InProgress` and secondary job status value `FeatureEngineering`.

This cell will take approximately 10 minutes to run.

In [24]:
```
%%time

job_description_response = automl.describe_auto_ml_job(job_name=auto_ml_j
job_status = job_description_response['AutoMLJobStatus']
job_sec_status = job_description_response['AutoMLJobSecondaryStatus']
print(job_status)
print(job_sec_status)
if job_status not in ('Stopped', 'Failed'):
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    while job_status == None and job_sec_status == None: # Replace all No
    ### END SOLUTION - DO NOT delete this comment for grading purposes
        job_description_response = automl.describe_auto_ml_job(job_name=a
        job_status = job_description_response['AutoMLJobStatus']
        job_sec_status = job_description_response['AutoMLJobSecondaryStat
        print(job_status, job_sec_status)
        time.sleep(5)
    print('[OK] Feature engineering phase completed.\n')

print(json.dumps(job_description_response, indent=4, sort_keys=True, defa
```

```
InProgress
GeneratingModelInsightsReport
[OK] Feature engineering phase completed.

{
    "AutoMLJobArn": "arn:aws:sagemaker:us-east-1:917875584351:automl-job/
automl-dm-1661262662",
    "AutoMLJobArtifacts": {
        "CandidateDefinitionNotebookLocation": "s3://sagemaker-us-east-1-
917875584351/autopilot/automl-dm-1661262662/sagemaker-automl-candidates/a
utoml-dm-1661262662-pr-1-61719c565070432bb34d947dd84accb25aafb/notebooks/
SageMakerAutopilotCandidateDefinitionNotebook.ipynb",
        "DataExplorationNotebookLocation": "s3://sagemaker-us-east-1-9178
75584351/autopilot/automl-dm-1661262662/sagemaker-automl-candidates/autom
l-dm-1661262662-pr-1-61719c565070432bb34d947dd84accb25aafb/notebooks/Sage
MakerAutopilotDataExplorationNotebook.ipynb"
    },
    "AutoMLJobConfig": {
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": 7200,
            "MaxCandidates": 3,
            "MaxRuntimePerTrainingJobInSeconds": 1200
        },
        "SecurityConfig": {
            "EnableInterContainerTrafficEncryption": false
        }
```

```
    },
    "AutoMLJobName": "automl-dm-1661262662",
    "AutoMLJobSecondaryStatus": "GeneratingModelInsightsReport",
    "AutoMLJobStatus": "InProgress",
    "BestCandidate": {
        "CandidateName": "automl-dm-1661262662dOwiVeHvxc2g-002-105c69d9",
        "CandidateProperties": {
            "CandidateArtifactLocations": {
                "Explainability": "s3://sagemaker-us-east-1-917875584351/
autopilot/automl-dm-1661262662/documentation/explainability/output"
            },
            "CandidateMetrics": [
                {
                    "MetricName": "Accuracy",
                    "Set": "Validation",
                    "StandardMetricName": "Accuracy",
                    "Value": 0.6198300123214722
                },
                {
                    "MetricName": "PrecisionMacro",
                    "Set": "Validation",
                    "StandardMetricName": "PrecisionMacro",
                    "Value": 0.6221399903297424
                },
                {
                    "MetricName": "BalancedAccuracy",
                    "Set": "Validation",
                    "StandardMetricName": "BalancedAccuracy",
                    "Value": 0.6198300123214722
                },
                {
                    "MetricName": "F1macro",
                    "Set": "Validation",
                    "StandardMetricName": "F1macro",
                    "Value": 0.6208199858665466
                },
                {
                    "MetricName": "RecallMacro",
                    "Set": "Validation",
                    "StandardMetricName": "RecallMacro",
                    "Value": 0.6198300123214722
                }
            ]
        },
        "CandidateStatus": "Completed",
        "CandidateSteps": [
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:processing-job/automl-dm-1661262662-db-1-53de988d6c0b4adb93322efa4a1
6ef99db41a",
                "CandidateStepName": "automl-dm-1661262662-db-1-53de988d6
c0b4adb93322efa4a16ef99db41a",
                "CandidateStepType": "AWS::SageMaker::ProcessingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662-dpp2-1-10806205885144c29a1074a63f7
be09e58d",
                "CandidateStepName": "automl-dm-1661262662-dpp2-1-1080620
5885144c29a1074a63f7be09e58d",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
```

```
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:transform-job/automl-dm-1661262662-dpp2-rpb-1-0efdb52b7b104271a6cdc7
231535111",
                "CandidateStepName": "automl-dm-1661262662-dpp2-rpb-1-0ef
db52b7b104271a6cdc7231535111",
                "CandidateStepType": "AWS::SageMaker::TransformJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662dowivehvxc2g-002-105c69d9",
                "CandidateStepName": "automl-dm-1661262662dOwiVeHvxc2g-00
2-105c69d9",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            }
        ],
        "CreationTime": "2022-08-23 14:18:46+00:00",
        "EndTime": "2022-08-23 14:20:54+00:00",
        "FinalAutoMLJobObjectiveMetric": {
            "MetricName": "validation:accuracy",
            "Value": 0.6198300123214722
        },
        "InferenceContainers": [
            {
                "Environment": {
                    "AUTOML_SPARSE_ENCODE_RECORDIO_PROTOBUF": "1",
                    "AUTOML_TRANSFORM_MODE": "feature-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "application/
x-recordio-protobuf",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp2-1-10806205885144c29a1074a63f7be09e58d/output/model.tar.gz"
            },
            {
                "Environment": {
                    "MAX_CONTENT_LENGTH": "20971520",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,probabilities"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-xgboost:1.3-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/tuning/automl-dm--dpp2-xgb/automl-dm-1661262
662dOwiVeHvxc2g-002-105c69d9/output/model.tar.gz"
            },
            {
                "Environment": {
                    "AUTOML_TRANSFORM_MODE": "inverse-label-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_INPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,labels,probabilities",
```

```
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp2-1-10806205885144c29a1074a63f7be09e58d/output/model.tar.gz"
            }
        ],
        "LastModifiedTime": "2022-08-23 14:31:44.722000+00:00",
        "ObjectiveStatus": "Succeeded"
    },
    "CreationTime": "2022-08-23 13:58:13.486000+00:00",
    "GenerateCandidateDefinitionsOnly": false,
    "InputDataConfig": [
        {
            "ChannelType": "training",
            "ContentType": "text/csv;header=present",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://sagemaker-us-east-1-917875584351/autop
ilot/data/womens_clothing_ecommerce_reviews_balanced_for_autopilot.csv"
                }
            },
            "TargetAttributeName": "sentiment"
        }
    ],
    "LastModifiedTime": "2022-08-23 14:43:56.320000+00:00",
    "OutputDataConfig": {
        "S3OutputPath": "s3://sagemaker-us-east-1-917875584351/autopilot"
    },
    "ResolvedAttributes": {
        "AutoMLJobObjective": {
            "MetricName": "Accuracy"
        },
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": 7200,
            "MaxCandidates": 3,
            "MaxRuntimePerTrainingJobInSeconds": 1200
        },
        "ProblemType": "MulticlassClassification"
    },
    "ResponseMetadata": {
        "HTTPHeaders": {
            "content-length": "5789",
            "content-type": "application/x-amz-json-1.1",
            "date": "Tue, 23 Aug 2022 14:44:03 GMT",
            "x-amzn-requestid": "0f09f5d0-fd5e-498d-8ff2-95d4f954044e"
        },
        "HTTPStatusCode": 200,
        "RequestId": "0f09f5d0-fd5e-498d-8ff2-95d4f954044e",
        "RetryAttempts": 0
    },
    "RoleArn": "arn:aws:iam::917875584351:role/sagemaker-studio-vpc-firew
all-us-east-1-sagemaker-execution-role"
}
CPU times: user 14.5 ms, sys: 6.52 ms, total: 21 ms
Wall time: 230 ms
```

6. Model training and tuning

When you launched the Autopilot job, you requested that 3 model candidates are generated and compared. Therefore, you should see three (3) SageMaker training jobs below.

In [25]:
```
from IPython.core.display import display, HTML

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

Review hyper-parameter tuning jobs

6.1. Wait for training and tuning

Exercise 6

Check the completion of the model tuning step.

Instructions: Use status check scheme described above. Model tuning step can be identified with the (primary) job status value `InProgress` and secondary job status value `ModelTuning`.

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

In [26]:
```
%%time

job_description_response = automl.describe_auto_ml_job(job_name=auto_ml_j
job_status = job_description_response['AutoMLJobStatus']
job_sec_status = job_description_response['AutoMLJobSecondaryStatus']
print(job_status)
print(job_sec_status)
if job_status not in ('Stopped', 'Failed'):
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    while job_status == None and job_sec_status == None: # Replace all No
    ### END SOLUTION - DO NOT delete this comment for grading purposes
        job_description_response = automl.describe_auto_ml_job(job_name=a
        job_status = job_description_response['AutoMLJobStatus']
        job_sec_status = job_description_response['AutoMLJobSecondaryStat
        print(job_status, job_sec_status)
        time.sleep(5)
    print('[OK] Model tuning phase completed.\n')

print(json.dumps(job_description_response, indent=4, sort_keys=True, defa
```

```
InProgress
GeneratingModelInsightsReport
[OK] Model tuning phase completed.

{
    "AutoMLJobArn": "arn:aws:sagemaker:us-east-1:917875584351:automl-job/
```

automl-dm-1661262662",
    "AutoMLJobArtifacts": {
        "CandidateDefinitionNotebookLocation": "s3://sagemaker-us-east-1-917875584351/autopilot/automl-dm-1661262662/sagemaker-automl-candidates/automl-dm-1661262662-pr-1-61719c565070432bb34d947dd84accb25aafb/notebooks/SageMakerAutopilotCandidateDefinitionNotebook.ipynb",
        "DataExplorationNotebookLocation": "s3://sagemaker-us-east-1-917875584351/autopilot/automl-dm-1661262662/sagemaker-automl-candidates/automl-dm-1661262662-pr-1-61719c565070432bb34d947dd84accb25aafb/notebooks/SageMakerAutopilotDataExplorationNotebook.ipynb"
    },
    "AutoMLJobConfig": {
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": 7200,
            "MaxCandidates": 3,
            "MaxRuntimePerTrainingJobInSeconds": 1200
        },
        "SecurityConfig": {
            "EnableInterContainerTrafficEncryption": false
        }
    },
    "AutoMLJobName": "automl-dm-1661262662",
    "AutoMLJobSecondaryStatus": "GeneratingModelInsightsReport",
    "AutoMLJobStatus": "InProgress",
    "BestCandidate": {
        "CandidateName": "automl-dm-1661262662dOwiVeHvxc2g-002-105c69d9",
        "CandidateProperties": {
            "CandidateArtifactLocations": {
                "Explainability": "s3://sagemaker-us-east-1-917875584351/autopilot/automl-dm-1661262662/documentation/explainability/output"
            },
            "CandidateMetrics": [
                {
                    "MetricName": "Accuracy",
                    "Set": "Validation",
                    "StandardMetricName": "Accuracy",
                    "Value": 0.6198300123214722
                },
                {
                    "MetricName": "PrecisionMacro",
                    "Set": "Validation",
                    "StandardMetricName": "PrecisionMacro",
                    "Value": 0.6221399903297424
                },
                {
                    "MetricName": "BalancedAccuracy",
                    "Set": "Validation",
                    "StandardMetricName": "BalancedAccuracy",
                    "Value": 0.6198300123214722
                },
                {
                    "MetricName": "F1macro",
                    "Set": "Validation",
                    "StandardMetricName": "F1macro",
                    "Value": 0.6208199858665466
                },
                {
                    "MetricName": "RecallMacro",
                    "Set": "Validation",
                    "StandardMetricName": "RecallMacro",
                    "Value": 0.6198300123214722

```
                }
            ]
        },
        "CandidateStatus": "Completed",
        "CandidateSteps": [
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:processing-job/automl-dm-1661262662-db-1-53de988d6c0b4adb93322efa4a1
6ef99db41a",
                "CandidateStepName": "automl-dm-1661262662-db-1-53de988d6
c0b4adb93322efa4a16ef99db41a",
                "CandidateStepType": "AWS::SageMaker::ProcessingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662-dpp2-1-10806205885144c29a1074a63f7
be09e58d",
                "CandidateStepName": "automl-dm-1661262662-dpp2-1-1080620
5885144c29a1074a63f7be09e58d",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:transform-job/automl-dm-1661262662-dpp2-rpb-1-0efdb52b7b104271a6cdc7
231535111",
                "CandidateStepName": "automl-dm-1661262662-dpp2-rpb-1-0ef
db52b7b104271a6cdc7231535111",
                "CandidateStepType": "AWS::SageMaker::TransformJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662dowivehvxc2g-002-105c69d9",
                "CandidateStepName": "automl-dm-1661262662dOwiVeHvxc2g-00
2-105c69d9",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            }
        ],
        "CreationTime": "2022-08-23 14:18:46+00:00",
        "EndTime": "2022-08-23 14:20:54+00:00",
        "FinalAutoMLJobObjectiveMetric": {
            "MetricName": "validation:accuracy",
            "Value": 0.6198300123214722
        },
        "InferenceContainers": [
            {
                "Environment": {
                    "AUTOML_SPARSE_ENCODE_RECORDIO_PROTOBUF": "1",
                    "AUTOML_TRANSFORM_MODE": "feature-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "application/
x-recordio-protobuf",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp2-1-10806205885144c29a1074a63f7be09e58d/output/model.tar.gz"
            },
            {
                "Environment": {
```

```
                    "MAX_CONTENT_LENGTH": "20971520",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,probabilities"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-xgboost:1.3-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/tuning/automl-dm--dpp2-xgb/automl-dm-1661262
662dOwiVeHvxc2g-002-105c69d9/output/model.tar.gz"
            },
            {
                "Environment": {
                    "AUTOML_TRANSFORM_MODE": "inverse-label-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_INPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,labels,probabilities",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp2-1-10806205885144c29a1074a63f7be09e58d/output/model.tar.gz"
            }
        ],
        "LastModifiedTime": "2022-08-23 14:31:44.722000+00:00",
        "ObjectiveStatus": "Succeeded"
    },
    "CreationTime": "2022-08-23 13:58:13.486000+00:00",
    "GenerateCandidateDefinitionsOnly": false,
    "InputDataConfig": [
        {
            "ChannelType": "training",
            "ContentType": "text/csv;header=present",
            "DataSource": {
                "S3DataSource": {
                    "S3DataType": "S3Prefix",
                    "S3Uri": "s3://sagemaker-us-east-1-917875584351/autop
ilot/data/womens_clothing_ecommerce_reviews_balanced_for_autopilot.csv"
                }
            },
            "TargetAttributeName": "sentiment"
        }
    ],
    "LastModifiedTime": "2022-08-23 14:44:13.394000+00:00",
    "OutputDataConfig": {
        "S3OutputPath": "s3://sagemaker-us-east-1-917875584351/autopilot"
    },
    "ResolvedAttributes": {
        "AutoMLJobObjective": {
            "MetricName": "Accuracy"
        },
        "CompletionCriteria": {
            "MaxAutoMLJobRuntimeInSeconds": 7200,
            "MaxCandidates": 3,
            "MaxRuntimePerTrainingJobInSeconds": 1200
```

```
        },
        "ProblemType": "MulticlassClassification"
    },
    "ResponseMetadata": {
        "HTTPHeaders": {
            "content-length": "5790",
            "content-type": "application/x-amz-json-1.1",
            "date": "Tue, 23 Aug 2022 14:44:14 GMT",
            "x-amzn-requestid": "90953695-43a4-4a73-a07c-6f250039c0ca"
        },
        "HTTPStatusCode": 200,
        "RequestId": "90953695-43a4-4a73-a07c-6f250039c0ca",
        "RetryAttempts": 0
    },
    "RoleArn": "arn:aws:iam::917875584351:role/sagemaker-studio-vpc-firew
all-us-east-1-sagemaker-execution-role"
}
CPU times: user 18.6 ms, sys: 3.32 ms, total: 21.9 ms
Wall time: 224 ms
```

Please wait until ^^ Autopilot ^^ completes above

Finally, you can check the completion of the Autopilot job looking for the `Completed` job status.

In [27]:

```
%%time

from pprint import pprint

job_description_response = automl.describe_auto_ml_job(job_name=auto_ml_j
pprint(job_description_response)
job_status = job_description_response['AutoMLJobStatus']
job_sec_status = job_description_response['AutoMLJobSecondaryStatus']
print('Job status:  {}'.format(job_status))
print('Secondary job status:  {}'.format(job_sec_status))
if job_status not in ('Stopped', 'Failed'):
    while job_status not in ('Completed'):
        job_description_response = automl.describe_auto_ml_job(job_name=a
        job_status = job_description_response['AutoMLJobStatus']
        job_sec_status = job_description_response['AutoMLJobSecondaryStat
        print('Job status:  {}'.format(job_status))
        print('Secondary job status:  {}'.format(job_sec_status))
        time.sleep(10)
    print('[OK] Autopilot job completed.\n')
else:
    print('Job status: {}'.format(job_status))
    print('Secondary job status: {}'.format(job_status))
```

```
{'AutoMLJobArn': 'arn:aws:sagemaker:us-east-1:917875584351:automl-job/aut
oml-dm-1661262662',
 'AutoMLJobArtifacts': {'CandidateDefinitionNotebookLocation': 's3://sage
maker-us-east-1-917875584351/autopilot/automl-dm-1661262662/sagemaker-aut
oml-candidates/automl-dm-1661262662-pr-1-61719c565070432bb34d947dd84accb2
5aafb/notebooks/SageMakerAutopilotCandidateDefinitionNotebook.ipynb',
                        'DataExplorationNotebookLocation': 's3://sagemake
r-us-east-1-917875584351/autopilot/automl-dm-1661262662/sagemaker-automl-
candidates/automl-dm-1661262662-pr-1-61719c565070432bb34d947dd84accb25aaf
b/notebooks/SageMakerAutopilotDataExplorationNotebook.ipynb'},
 'AutoMLJobConfig': {'CompletionCriteria': {'MaxAutoMLJobRuntimeInSeconds
': 7200,
```

```
                                                 'MaxCandidates': 3,
                                                 'MaxRuntimePerTrainingJobInSe
conds': 1200},
                      'SecurityConfig': {'EnableInterContainerTrafficEncry
ption': False}},
 'AutoMLJobName': 'automl-dm-1661262662',
 'AutoMLJobSecondaryStatus': 'GeneratingModelInsightsReport',
 'AutoMLJobStatus': 'InProgress',
 'BestCandidate': {'CandidateName': 'automl-dm-1661262662dOwiVeHvxc2g-002
-105c69d9',
                   'CandidateProperties': {'CandidateArtifactLocations':
{'Explainability': 's3://sagemaker-us-east-1-917875584351/autopilot/autom
l-dm-1661262662/documentation/explainability/output'},
                                           'CandidateMetrics': [{'MetricN
ame': 'Accuracy',
                                                                 'Set': '
Validation',
                                                                 'Standar
dMetricName': 'Accuracy',
                                                                 'Value':
0.6198300123214722},
                                                                {'MetricN
ame': 'PrecisionMacro',
                                                                 'Set': '
Validation',
                                                                 'Standar
dMetricName': 'PrecisionMacro',
                                                                 'Value':
0.6221399903297424},
                                                                {'MetricN
ame': 'BalancedAccuracy',
                                                                 'Set': '
Validation',
                                                                 'Standar
dMetricName': 'BalancedAccuracy',
                                                                 'Value':
0.6198300123214722},
                                                                {'MetricN
ame': 'F1macro',
                                                                 'Set': '
Validation',
                                                                 'Standar
dMetricName': 'F1macro',
                                                                 'Value':
0.6208199858665466},
                                                                {'MetricN
ame': 'RecallMacro',
                                                                 'Set': '
Validation',
                                                                 'Standar
dMetricName': 'RecallMacro',
                                                                 'Value':
0.6198300123214722}]},
                   'CandidateStatus': 'Completed',
                   'CandidateSteps': [{'CandidateStepArn': 'arn:aws:sagem
aker:us-east-1:917875584351:processing-job/automl-dm-1661262662-db-1-53de
988d6c0b4adb93322efa4a16ef99db41a',
                                       'CandidateStepName': 'automl-dm-16
61262662-db-1-53de988d6c0b4adb93322efa4a16ef99db41a',
                                       'CandidateStepType': 'AWS::SageMak
er::ProcessingJob'},
```

```
                                        {'CandidateStepArn': 'arn:aws:sagem
aker:us-east-1:917875584351:training-job/automl-dm-1661262662-dpp2-1-1080
6205885144c29a1074a63f7be09e58d',
                                         'CandidateStepName': 'automl-dm-16
61262662-dpp2-1-10806205885144c29a1074a63f7be09e58d',
                                         'CandidateStepType': 'AWS::SageMak
er::TrainingJob'},
                                        {'CandidateStepArn': 'arn:aws:sagem
aker:us-east-1:917875584351:transform-job/automl-dm-1661262662-dpp2-rpb-1
-0efdb52b7b104271a6cdc7231535111',
                                         'CandidateStepName': 'automl-dm-16
61262662-dpp2-rpb-1-0efdb52b7b104271a6cdc7231535111',
                                         'CandidateStepType': 'AWS::SageMak
er::TransformJob'},
                                        {'CandidateStepArn': 'arn:aws:sagem
aker:us-east-1:917875584351:training-job/automl-dm-1661262662dowivehvxc2g
-002-105c69d9',
                                         'CandidateStepName': 'automl-dm-16
61262662dOwiVeHvxc2g-002-105c69d9',
                                         'CandidateStepType': 'AWS::SageMak
er::TrainingJob'}],
                    'CreationTime': datetime.datetime(2022, 8, 23, 14, 18,
46, tzinfo=tzlocal()),
                    'EndTime': datetime.datetime(2022, 8, 23, 14, 20, 54,
tzinfo=tzlocal()),
                    'FinalAutoMLJobObjectiveMetric': {'MetricName': 'valid
ation:accuracy',
                                                      'Value': 0.619830012
3214722},
                    'InferenceContainers': [{'Environment': {'AUTOML_SPARS
E_ENCODE_RECORDIO_PROTOBUF': '1',
                                                             'AUTOML_TRANS
FORM_MODE': 'feature-transform',
                                                             'SAGEMAKER_DE
FAULT_INVOCATIONS_ACCEPT': 'application/x-recordio-protobuf',
                                                             'SAGEMAKER_PR
OGRAM': 'sagemaker_serve',
                                                             'SAGEMAKER_SU
BMIT_DIRECTORY': '/opt/ml/model/code'},
                                                 'Image': '683313688378.dkr.ec
r.us-east-1.amazonaws.com/sagemaker-sklearn-automl:2.5-1-cpu-py3',
                                                 'ModelDataUrl': 's3://sagemak
er-us-east-1-917875584351/autopilot/automl-dm-1661262662/data-processor-m
odels/automl-dm-1661262662-dpp2-1-10806205885144c29a1074a63f7be09e58d/out
put/model.tar.gz'},
                                                {'Environment': {'MAX_CONTENT_
LENGTH': '20971520',
                                                             'SAGEMAKER_DE
FAULT_INVOCATIONS_ACCEPT': 'text/csv',
                                                             'SAGEMAKER_IN
FERENCE_OUTPUT': 'predicted_label',
                                                             'SAGEMAKER_IN
FERENCE_SUPPORTED': 'predicted_label,probability,probabilities'},
                                                 'Image': '683313688378.dkr.ec
r.us-east-1.amazonaws.com/sagemaker-xgboost:1.3-1-cpu-py3',
                                                 'ModelDataUrl': 's3://sagemak
er-us-east-1-917875584351/autopilot/automl-dm-1661262662/tuning/automl-dm
--dpp2-xgb/automl-dm-1661262662dOwiVeHvxc2g-002-105c69d9/output/model.tar
.gz'},
                                                {'Environment': {'AUTOML_TRANS
FORM_MODE': 'inverse-label-transform',
```

```
                                                                 'SAGEMAKER_DE
FAULT_INVOCATIONS_ACCEPT': 'text/csv',
                                                                 'SAGEMAKER_IN
FERENCE_INPUT': 'predicted_label',
                                                                 'SAGEMAKER_IN
FERENCE_OUTPUT': 'predicted_label',
                                                                 'SAGEMAKER_IN
FERENCE_SUPPORTED': 'predicted_label,probability,labels,probabilities',
                                                                 'SAGEMAKER_PR
OGRAM': 'sagemaker_serve',
                                                                 'SAGEMAKER_SU
BMIT_DIRECTORY': '/opt/ml/model/code'},
                                                 'Image': '683313688378.dkr.ec
r.us-east-1.amazonaws.com/sagemaker-sklearn-automl:2.5-1-cpu-py3',
                                                 'ModelDataUrl': 's3://sagemak
er-us-east-1-917875584351/autopilot/automl-dm-1661262662/data-processor-m
odels/automl-dm-1661262662-dpp2-1-10806205885144c29a1074a63f7be09e58d/out
put/model.tar.gz'}],
                    'LastModifiedTime': datetime.datetime(2022, 8, 23, 14,
31, 44, 722000, tzinfo=tzlocal()),
                    'ObjectiveStatus': 'Succeeded'},
 'CreationTime': datetime.datetime(2022, 8, 23, 13, 58, 13, 486000, tzinf
o=tzlocal()),
 'GenerateCandidateDefinitionsOnly': False,
 'InputDataConfig': [{'ChannelType': 'training',
                      'ContentType': 'text/csv;header=present',
                      'DataSource': {'S3DataSource': {'S3DataType': 'S3Pr
efix',
                                                      'S3Uri': 's3://sage
maker-us-east-1-917875584351/autopilot/data/womens_clothing_ecommerce_rev
iews_balanced_for_autopilot.csv'}},
                      'TargetAttributeName': 'sentiment'}],
 'LastModifiedTime': datetime.datetime(2022, 8, 23, 14, 44, 13, 394000, t
zinfo=tzlocal()),
 'OutputDataConfig': {'S3OutputPath': 's3://sagemaker-us-east-1-917875584
351/autopilot'},
 'ResolvedAttributes': {'AutoMLJobObjective': {'MetricName': 'Accuracy'},
                        'CompletionCriteria': {'MaxAutoMLJobRuntimeInSeco
nds': 7200,
                                               'MaxCandidates': 3,
                                               'MaxRuntimePerTrainingJobI
nSeconds': 1200},
                        'ProblemType': 'MulticlassClassification'},
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '5790',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Tue, 23 Aug 2022 14:44:17
GMT',
                                      'x-amzn-requestid': '524ce9df-c673-
44ab-9c9c-869cb7aa8a51'},
                      'HTTPStatusCode': 200,
                      'RequestId': '524ce9df-c673-44ab-9c9c-869cb7aa8a51'
,
                      'RetryAttempts': 0},
 'RoleArn': 'arn:aws:iam::917875584351:role/sagemaker-studio-vpc-firewall
-us-east-1-sagemaker-execution-role'}
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
```

```
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  InProgress
Secondary job status:  GeneratingModelInsightsReport
Job status:  Completed
Secondary job status:  Completed
[OK] Autopilot job completed.

CPU times: user 366 ms, sys: 45.5 ms, total: 411 ms
Wall time: 4min 24s
```

Before moving to the next section make sure the status above indicates `Autopilot job completed`.

6.2. Compare model candidates

Once model tuning is complete, you can view all the candidates (pipeline evaluations with different hyperparameter combinations) that were explored by AutoML and sort them by their final performance metric.

Exercise 7

List candidates generated by Autopilot sorted by accuracy from highest to lowest.

Instructions: Use `list_candidates` function passing the Autopilot job name `auto_ml_job_name` with the accuracy field `FinalObjectiveMetricValue`. It returns the list of candidates with the information about them.

```
candidates = automl.list_candidates(
    job_name=..., # Autopilot job name
    sort_by='...' # accuracy field name
)
```

In [29]:
```
candidates = automl.list_candidates(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    job_name=auto_ml_job_name, # Replace None
    sort_by='FinalObjectiveMetricValue' # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
)
```

You can review the response syntax and response elements of the function `list_candidates` in the **documentation**. Now let's put the candidate existence check into the loop:

In [30]:
```
while candidates == []:
    candidates = automl.list_candidates(job_name=auto_ml_job_name)
    print('[INFO] Autopilot job is generating the candidates. Please wait
    time.sleep(10)

print('[OK] Candidates generated.')
```

```
[OK] Candidates generated.
```

The information about each of the candidates is in the dictionary with the following keys:

In [31]:
```
print(candidates[0].keys())
```

```
dict_keys(['CandidateName', 'FinalAutoMLJobObjectiveMetric', 'ObjectiveSt
atus', 'CandidateSteps', 'CandidateStatus', 'InferenceContainers', 'Creat
ionTime', 'EndTime', 'LastModifiedTime', 'CandidateProperties'])
```

`CandidateName` contains the candidate name and the `FinalAutoMLJobObjectiveMetric` element contains the metric information which can be used to identify the best candidate later. Let's check that they were generated.

In [32]:
```
while 'CandidateName' not in candidates[0]:
    candidates = automl.list_candidates(job_name=auto_ml_job_name)
    print('[INFO] Autopilot job is generating CandidateName. Please wait.
    sleep(10)

print('[OK] CandidateName generated.')
```

```
[OK] CandidateName generated.
```

In [33]:
```
while 'FinalAutoMLJobObjectiveMetric' not in candidates[0]:
    candidates = automl.list_candidates(job_name=auto_ml_job_name)
    print('[INFO] Autopilot job is generating FinalAutoMLJobObjectiveMetr
    sleep(10)

print('[OK] FinalAutoMLJobObjectiveMetric generated.')
```

```
[OK] FinalAutoMLJobObjectiveMetric generated.
```

In [34]:
```
print(json.dumps(candidates, indent=4, sort_keys=True, default=str))
```


```
[
    {
        "CandidateName": "automl-dm-1661262662dOwiVeHvxc2g-002-105c69d9",
        "CandidateProperties": {
            "CandidateArtifactLocations": {
                "Explainability": "s3://sagemaker-us-east-1-917875584351/
autopilot/automl-dm-1661262662/documentation/explainability/output",
                "ModelInsights": "s3://sagemaker-us-east-1-917875584351/a
utopilot/automl-dm-1661262662/documentation/model_monitor/output"
            },
            "CandidateMetrics": [
                {
                    "MetricName": "Accuracy",
                    "Set": "Validation",
                    "StandardMetricName": "Accuracy",
                    "Value": 0.6198300123214722
                },
                {
                    "MetricName": "PrecisionMacro",
                    "Set": "Validation",
                    "StandardMetricName": "PrecisionMacro",
                    "Value": 0.6221399903297424
                },
                {
                    "MetricName": "BalancedAccuracy",
                    "Set": "Validation",
                    "StandardMetricName": "BalancedAccuracy",
                    "Value": 0.6198300123214722
                },
                {
                    "MetricName": "F1macro",
                    "Set": "Validation",
                    "StandardMetricName": "F1macro",
                    "Value": 0.6208199858665466
```

```
                },
                {
                    "MetricName": "RecallMacro",
                    "Set": "Validation",
                    "StandardMetricName": "RecallMacro",
                    "Value": 0.6198300123214722
                }
            ]
        },
        "CandidateStatus": "Completed",
        "CandidateSteps": [
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:processing-job/automl-dm-1661262662-db-1-53de988d6c0b4adb93322efa4a1
6ef99db41a",
                "CandidateStepName": "automl-dm-1661262662-db-1-53de988d6
c0b4adb93322efa4a16ef99db41a",
                "CandidateStepType": "AWS::SageMaker::ProcessingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662-dpp2-1-10806205885144c29a1074a63f7
be09e58d",
                "CandidateStepName": "automl-dm-1661262662-dpp2-1-1080620
5885144c29a1074a63f7be09e58d",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:transform-job/automl-dm-1661262662-dpp2-rpb-1-0efdb52b7b104271a6cdc7
231535111",
                "CandidateStepName": "automl-dm-1661262662-dpp2-rpb-1-0ef
db52b7b104271a6cdc7231535111",
                "CandidateStepType": "AWS::SageMaker::TransformJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662dowivehvxc2g-002-105c69d9",
                "CandidateStepName": "automl-dm-1661262662dOwiVeHvxc2g-00
2-105c69d9",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            }
        ],
        "CreationTime": "2022-08-23 14:18:46+00:00",
        "EndTime": "2022-08-23 14:20:54+00:00",
        "FinalAutoMLJobObjectiveMetric": {
            "MetricName": "validation:accuracy",
            "Value": 0.6198300123214722
        },
        "InferenceContainers": [
            {
                "Environment": {
                    "AUTOML_SPARSE_ENCODE_RECORDIO_PROTOBUF": "1",
                    "AUTOML_TRANSFORM_MODE": "feature-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "application/
x-recordio-protobuf",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
```

```
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp2-1-10806205885144c29a1074a63f7be09e58d/output/model.tar.gz"
            },
            {
                "Environment": {
                    "MAX_CONTENT_LENGTH": "20971520",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,probabilities"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-xgboost:1.3-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/tuning/automl-dm--dpp2-xgb/automl-dm-1661262
662dOwiVeHvxc2g-002-105c69d9/output/model.tar.gz"
            },
            {
                "Environment": {
                    "AUTOML_TRANSFORM_MODE": "inverse-label-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_INPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,labels,probabilities",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp2-1-10806205885144c29a1074a63f7be09e58d/output/model.tar.gz"
            }
        ],
        "LastModifiedTime": "2022-08-23 14:31:44.722000+00:00",
        "ObjectiveStatus": "Succeeded"
    },
    {
        "CandidateName": "automl-dm-1661262662dOwiVeHvxc2g-001-b0a3f023",
        "CandidateProperties": {
            "CandidateMetrics": [
                {
                    "MetricName": "Accuracy",
                    "Set": "Validation",
                    "StandardMetricName": "Accuracy",
                    "Value": 0.4041700065135956
                },
                {
                    "MetricName": "PrecisionMacro",
                    "Set": "Validation",
                    "StandardMetricName": "PrecisionMacro",
                    "Value": 0.4039599895477295
                },
                {
                    "MetricName": "BalancedAccuracy",
                    "Set": "Validation",
                    "StandardMetricName": "BalancedAccuracy",
                    "Value": 0.4041700065135956
                },
```

```
                {
                    "MetricName": "F1macro",
                    "Set": "Validation",
                    "StandardMetricName": "F1macro",
                    "Value": 0.4038900136947632
                },
                {
                    "MetricName": "RecallMacro",
                    "Set": "Validation",
                    "StandardMetricName": "RecallMacro",
                    "Value": 0.4041700065135956
                }
            ]
        },
        "CandidateStatus": "Completed",
        "CandidateSteps": [
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:processing-job/automl-dm-1661262662-db-1-53de988d6c0b4adb93322efa4a1
6ef99db41a",
                "CandidateStepName": "automl-dm-1661262662-db-1-53de988d6
c0b4adb93322efa4a16ef99db41a",
                "CandidateStepType": "AWS::SageMaker::ProcessingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662-dpp1-1-6de12eef194d4ed69d1f49b982c
f26ad8f7",
                "CandidateStepName": "automl-dm-1661262662-dpp1-1-6de12ee
f194d4ed69d1f49b982cf26ad8f7",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:transform-job/automl-dm-1661262662-dpp1-csv-1-309d5a78731a47688872a6
31d7a6174",
                "CandidateStepName": "automl-dm-1661262662-dpp1-csv-1-309
d5a78731a47688872a631d7a6174",
                "CandidateStepType": "AWS::SageMaker::TransformJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662dowivehvxc2g-001-b0a3f023",
                "CandidateStepName": "automl-dm-1661262662dOwiVeHvxc2g-00
1-b0a3f023",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            }
        ],
        "CreationTime": "2022-08-23 14:18:48+00:00",
        "EndTime": "2022-08-23 14:21:56+00:00",
        "FinalAutoMLJobObjectiveMetric": {
            "MetricName": "validation:accuracy",
            "Value": 0.4041700065135956
        },
        "InferenceContainers": [
            {
                "Environment": {
                    "AUTOML_TRANSFORM_MODE": "feature-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "application/
x-recordio-protobuf",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
```

```
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp1-1-6de12eef194d4ed69d1f49b982cf26ad8f7/output/model.tar.gz"
            },
            {
                "Environment": {
                    "MAX_CONTENT_LENGTH": "20971520",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,probabilities"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-xgboost:1.3-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/tuning/automl-dm--dpp1-xgb/automl-dm-1661262
662dOwiVeHvxc2g-001-b0a3f023/output/model.tar.gz"
            },
            {
                "Environment": {
                    "AUTOML_TRANSFORM_MODE": "inverse-label-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_INPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,labels,probabilities",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp1-1-6de12eef194d4ed69d1f49b982cf26ad8f7/output/model.tar.gz"
            }
        ],
        "LastModifiedTime": "2022-08-23 14:31:44.650000+00:00",
        "ObjectiveStatus": "Succeeded"
    },
    {
        "CandidateName": "automl-dm-1661262662dOwiVeHvxc2g-003-ca78fb67",
        "CandidateProperties": {
            "CandidateMetrics": [
                {
                    "MetricName": "Accuracy",
                    "Set": "Validation",
                    "StandardMetricName": "Accuracy",
                    "Value": 0.40327998995780945
                },
                {
                    "MetricName": "PrecisionMacro",
                    "Set": "Validation",
                    "StandardMetricName": "PrecisionMacro",
                    "Value": 0.40283000469207764
                },
                {
                    "MetricName": "BalancedAccuracy",
```

```
                    "Set": "Validation",
                    "StandardMetricName": "BalancedAccuracy",
                    "Value": 0.40327998995780945
                },
                {
                    "MetricName": "F1macro",
                    "Set": "Validation",
                    "StandardMetricName": "F1macro",
                    "Value": 0.40290001034736633
                },
                {
                    "MetricName": "RecallMacro",
                    "Set": "Validation",
                    "StandardMetricName": "RecallMacro",
                    "Value": 0.40327998995780945
                }
            ]
        },
        "CandidateStatus": "Completed",
        "CandidateSteps": [
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:processing-job/automl-dm-1661262662-db-1-53de988d6c0b4adb93322efa4a1
6ef99db41a",
                "CandidateStepName": "automl-dm-1661262662-db-1-53de988d6
c0b4adb93322efa4a16ef99db41a",
                "CandidateStepType": "AWS::SageMaker::ProcessingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662-dpp1-1-6de12eef194d4ed69d1f49b982c
f26ad8f7",
                "CandidateStepName": "automl-dm-1661262662-dpp1-1-6de12ee
f194d4ed69d1f49b982cf26ad8f7",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:transform-job/automl-dm-1661262662-dpp1-csv-1-309d5a78731a47688872a6
31d7a6174",
                "CandidateStepName": "automl-dm-1661262662-dpp1-csv-1-309
d5a78731a47688872a631d7a6174",
                "CandidateStepType": "AWS::SageMaker::TransformJob"
            },
            {
                "CandidateStepArn": "arn:aws:sagemaker:us-east-1:91787558
4351:training-job/automl-dm-1661262662dowivehvxc2g-003-ca78fb67",
                "CandidateStepName": "automl-dm-1661262662dOwiVeHvxc2g-00
3-ca78fb67",
                "CandidateStepType": "AWS::SageMaker::TrainingJob"
            }
        ],
        "CreationTime": "2022-08-23 14:18:52+00:00",
        "EndTime": "2022-08-23 14:30:47+00:00",
        "FinalAutoMLJobObjectiveMetric": {
            "MetricName": "validation:accuracy",
            "Value": 0.40327998995780945
        },
        "InferenceContainers": [
            {
                "Environment": {
```

```
                    "AUTOML_TRANSFORM_MODE": "feature-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "application/
x-recordio-protobuf",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp1-1-6de12eef194d4ed69d1f49b982cf26ad8f7/output/model.tar.gz"
            },
            {
                "Environment": {
                    "MAX_CONTENT_LENGTH": "20971520",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,probabilities"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-xgboost:1.3-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/tuning/automl-dm--dpp1-xgb/automl-dm-1661262
662dOwiVeHvxc2g-003-ca78fb67/output/model.tar.gz"
            },
            {
                "Environment": {
                    "AUTOML_TRANSFORM_MODE": "inverse-label-transform",
                    "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                    "SAGEMAKER_INFERENCE_INPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                    "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,pro
bability,labels,probabilities",
                    "SAGEMAKER_PROGRAM": "sagemaker_serve",
                    "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
                },
                "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sa
gemaker-sklearn-automl:2.5-1-cpu-py3",
                "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/au
topilot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-d
pp1-1-6de12eef194d4ed69d1f49b982cf26ad8f7/output/model.tar.gz"
            }
        ],
        "LastModifiedTime": "2022-08-23 14:31:44.650000+00:00",
        "ObjectiveStatus": "Succeeded"
    }
]
```


You can print the names of the candidates with their metric values:

In [35]:
```
print("metric " + str(candidates[0]['FinalAutoMLJobObjectiveMetric']['Met

for index, candidate in enumerate(candidates):
    print(str(index) + "  "
        + candidate['CandidateName'] + "  "
        + str(candidate['FinalAutoMLJobObjectiveMetric']['Value']))
```

```
metric validation:accuracy
0  automl-dm-1661262662dOwiVeHvxc2g-002-105c69d9  0.6198300123214722
1  automl-dm-1661262662dOwiVeHvxc2g-001-b0a3f023  0.4041700065135956
2  automl-dm-1661262662dOwiVeHvxc2g-003-ca78fb67  0.40327998995780945
```

6.3. Review best candidate

Now that you have successfully completed the Autopilot job on the dataset and visualized the trials, you can get the information about the best candidate model and review it.

Exercise 8

Get the information about the generated best candidate job.

Instructions: Use `best_candidate` function passing the Autopilot job name. This function will give an error if candidates have not been generated.

In [36]:
```
candidates = automl.list_candidates(job_name=auto_ml_job_name)

if candidates != []:
    best_candidate = automl.best_candidate(
        ### BEGIN SOLUTION - DO NOT delete this comment for grading purpo
        job_name=auto_ml_job_name # Replace None
        ### END SOLUTION - DO NOT delete this comment for grading purpose
    )
    print(json.dumps(best_candidate, indent=4, sort_keys=True, default=st
```

```
{
    "CandidateName": "automl-dm-1661262662dOwiVeHvxc2g-002-105c69d9",
    "CandidateProperties": {
        "CandidateArtifactLocations": {
            "Explainability": "s3://sagemaker-us-east-1-917875584351/auto
pilot/automl-dm-1661262662/documentation/explainability/output",
            "ModelInsights": "s3://sagemaker-us-east-1-917875584351/autop
ilot/automl-dm-1661262662/documentation/model_monitor/output"
        },
        "CandidateMetrics": [
            {
                "MetricName": "Accuracy",
                "Set": "Validation",
                "StandardMetricName": "Accuracy",
                "Value": 0.6198300123214722
            },
            {
                "MetricName": "PrecisionMacro",
                "Set": "Validation",
                "StandardMetricName": "PrecisionMacro",
                "Value": 0.6221399903297424
            },
            {
                "MetricName": "BalancedAccuracy",
                "Set": "Validation",
                "StandardMetricName": "BalancedAccuracy",
                "Value": 0.6198300123214722
```

```
            },
            {
                "MetricName": "F1macro",
                "Set": "Validation",
                "StandardMetricName": "F1macro",
                "Value": 0.6208199858665466
            },
            {
                "MetricName": "RecallMacro",
                "Set": "Validation",
                "StandardMetricName": "RecallMacro",
                "Value": 0.6198300123214722
            }
        ]
    },
    "CandidateStatus": "Completed",
    "CandidateSteps": [
        {
            "CandidateStepArn": "arn:aws:sagemaker:us-east-1:917875584351
:processing-job/automl-dm-1661262662-db-1-53de988d6c0b4adb93322efa4a16ef9
9db41a",
            "CandidateStepName": "automl-dm-1661262662-db-1-53de988d6c0b4
adb93322efa4a16ef99db41a",
            "CandidateStepType": "AWS::SageMaker::ProcessingJob"
        },
        {
            "CandidateStepArn": "arn:aws:sagemaker:us-east-1:917875584351
:training-job/automl-dm-1661262662-dpp2-1-10806205885144c29a1074a63f7be09
e58d",
            "CandidateStepName": "automl-dm-1661262662-dpp2-1-10806205885
144c29a1074a63f7be09e58d",
            "CandidateStepType": "AWS::SageMaker::TrainingJob"
        },
        {
            "CandidateStepArn": "arn:aws:sagemaker:us-east-1:917875584351
:transform-job/automl-dm-1661262662-dpp2-rpb-1-0efdb52b7b104271a6cdc72315
35111",
            "CandidateStepName": "automl-dm-1661262662-dpp2-rpb-1-0efdb52
b7b104271a6cdc7231535111",
            "CandidateStepType": "AWS::SageMaker::TransformJob"
        },
        {
            "CandidateStepArn": "arn:aws:sagemaker:us-east-1:917875584351
:training-job/automl-dm-1661262662dowivehvxc2g-002-105c69d9",
            "CandidateStepName": "automl-dm-1661262662dOwiVeHvxc2g-002-10
5c69d9",
            "CandidateStepType": "AWS::SageMaker::TrainingJob"
        }
    ],
    "CreationTime": "2022-08-23 14:18:46+00:00",
    "EndTime": "2022-08-23 14:20:54+00:00",
    "FinalAutoMLJobObjectiveMetric": {
        "MetricName": "validation:accuracy",
        "Value": 0.6198300123214722
    },
    "InferenceContainers": [
        {
            "Environment": {
                "AUTOML_SPARSE_ENCODE_RECORDIO_PROTOBUF": "1",
                "AUTOML_TRANSFORM_MODE": "feature-transform",
                "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "application/x-re
```

```
cordio-protobuf",
                "SAGEMAKER_PROGRAM": "sagemaker_serve",
                "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
            },
            "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sagema
ker-sklearn-automl:2.5-1-cpu-py3",
            "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/autopi
lot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-dpp2-
1-10806205885144c29a1074a63f7be09e58d/output/model.tar.gz"
        },
        {
            "Environment": {
                "MAX_CONTENT_LENGTH": "20971520",
                "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,probabi
lity,probabilities"
            },
            "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sagema
ker-xgboost:1.3-1-cpu-py3",
            "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/autopi
lot/automl-dm-1661262662/tuning/automl-dm--dpp2-xgb/automl-dm-1661262662d
OwiVeHvxc2g-002-105c69d9/output/model.tar.gz"
        },
        {
            "Environment": {
                "AUTOML_TRANSFORM_MODE": "inverse-label-transform",
                "SAGEMAKER_DEFAULT_INVOCATIONS_ACCEPT": "text/csv",
                "SAGEMAKER_INFERENCE_INPUT": "predicted_label",
                "SAGEMAKER_INFERENCE_OUTPUT": "predicted_label",
                "SAGEMAKER_INFERENCE_SUPPORTED": "predicted_label,probabi
lity,labels,probabilities",
                "SAGEMAKER_PROGRAM": "sagemaker_serve",
                "SAGEMAKER_SUBMIT_DIRECTORY": "/opt/ml/model/code"
            },
            "Image": "683313688378.dkr.ecr.us-east-1.amazonaws.com/sagema
ker-sklearn-automl:2.5-1-cpu-py3",
            "ModelDataUrl": "s3://sagemaker-us-east-1-917875584351/autopi
lot/automl-dm-1661262662/data-processor-models/automl-dm-1661262662-dpp2-
1-10806205885144c29a1074a63f7be09e58d/output/model.tar.gz"
        }
    ],
    "LastModifiedTime": "2022-08-23 14:31:44.722000+00:00",
    "ObjectiveStatus": "Succeeded"
}
```


Check the existence of the candidate name for the best candidate.

In [37]:
```
while 'CandidateName' not in best_candidate:
    best_candidate = automl.best_candidate(job_name=auto_ml_job_name)
    print('[INFO] Autopilot Job is generating BestCandidate CandidateName
    print(json.dumps(best_candidate, indent=4, sort_keys=True, default=st
    sleep(10)

print('[OK] BestCandidate CandidateName generated.')
```

```
[OK] BestCandidate CandidateName generated.
```

Check the existence of the metric value for the best candidate.

In [38]:
```
while 'FinalAutoMLJobObjectiveMetric' not in best_candidate:
    best_candidate = automl.best_candidate(job_name=auto_ml_job_name)
    print('[INFO] Autopilot Job is generating BestCandidate FinalAutoMLJo
    print(json.dumps(best_candidate, indent=4, sort_keys=True, default=st
    sleep(10)

print('[OK] BestCandidate FinalAutoMLJobObjectiveMetric generated.')
```

```
[OK] BestCandidate FinalAutoMLJobObjectiveMetric generated.
```

Print the information about the best candidate:

In [39]:
```
best_candidate_identifier = best_candidate['CandidateName']
print("Candidate name: " + best_candidate_identifier)
print("Metric name: " + best_candidate['FinalAutoMLJobObjectiveMetric']['
print("Metric value: " + str(best_candidate['FinalAutoMLJobObjectiveMetri
```

```
Candidate name: automl-dm-1661262662dOwiVeHvxc2g-002-105c69d9
Metric name: validation:accuracy
Metric value: 0.6198300123214722
```

7. Review all output in S3 bucket

You will see the artifacts generated by Autopilot including the following:

```
data-processor-models/        # "models" learned to
transform raw data into features
documentation/                # explainability and other
documentation about your model
preprocessed-data/            # data for train and
validation
sagemaker-automl-candidates/  # candidate models which
autopilot compares
transformed-data/             # candidate-specific data
for train and validation
tuning/                       # candidate-specific tuning
results
validations/                  # validation results
```

In [40]:
```
from IPython.core.display import display, HTML

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

Review all output in S3

8. Deploy and test best candidate model

8.1. Deploy best candidate model

While batch transformations are supported, you will deploy our model as a REST Endpoint in this example.

First, you need to customize the inference response. The inference containers generated by SageMaker Autopilot allow you to select the response content for predictions. By default the inference containers are configured to generate the `predicted_label`. But you can add `probability` into the list of inference response keys.

In [41]:
```
inference_response_keys = ['predicted_label', 'probability']
```

Now you will create a SageMaker endpoint from the best candidate generated by Autopilot. Wait for SageMaker to deploy the endpoint.

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

In [43]:
```
autopilot_model = automl.deploy(
    initial_instance_count=1,
    instance_type='ml.m5.large',
    candidate=best_candidate,
    inference_response_keys=inference_response_keys,
    predictor_cls=sagemaker.predictor.Predictor,
    serializer=sagemaker.serializers.JSONSerializer(),
    deserializer=sagemaker.deserializers.JSONDeserializer()
)

print('\nEndpoint name:  {}'.format(autopilot_model.endpoint_name))
```

```
-----------------!
Endpoint name:  sagemaker-sklearn-automl-2022-08-23-15-03-12-767
```

Please wait until the ^^ endpoint ^^ is deployed.

Review the SageMaker endpoint in the AWS console.

In [44]:
```
from IPython.core.display import display, HTML

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

Review SageMaker REST endpoint

8.2. Test the model

Invoke a few predictions for the actual reviews using the deployed endpoint.

In [45]:
```
#sm_runtime = boto3.client('sagemaker-runtime')

review_list = ['This product is great!',
               'OK, but not great.',
               'This is not the right product.']

for review in review_list:

    # remove commas from the review since we're passing the inputs as a C
    review = review.replace(",", "")

    response = sm_runtime.invoke_endpoint(
        EndpointName=autopilot_model.endpoint_name, # endpoint name
        ContentType='text/csv', # type of input data
        Accept='text/csv', # type of the inference in the response
        Body=review # review text
        )

    response_body=response['Body'].read().decode('utf-8').strip().split('

    print('Review: ', review, ' Predicated class: {}'.format(response_bod

print("(-1 = Negative, 0=Neutral, 1=Positive)")
```

```
Review:  This product is great!  Predicated class: 1
Review:  OK but not great.  Predicated class: 0
Review:  This is not the right product.  Predicated class: -1
(-1 = Negative, 0=Neutral, 1=Positive)
```

You used Amazon SageMaker Autopilot to automatically find the best model, hyper-parameters, and feature-engineering scripts for our dataset. Autopilot uses a uniquely-transparent approach to AutoML by generating re-usable Python scripts and notebooks.

Upload the notebook into S3 bucket for grading purposes.

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

In [ ]:
```
!aws s3 cp ./C1_W3_Assignment.ipynb s3://$bucket/C1_W3_Assignment_Learner
```

Please go to the main lab window and click on `Submit` button (see the `Finish the lab` section of the instructions).

In [ ]:

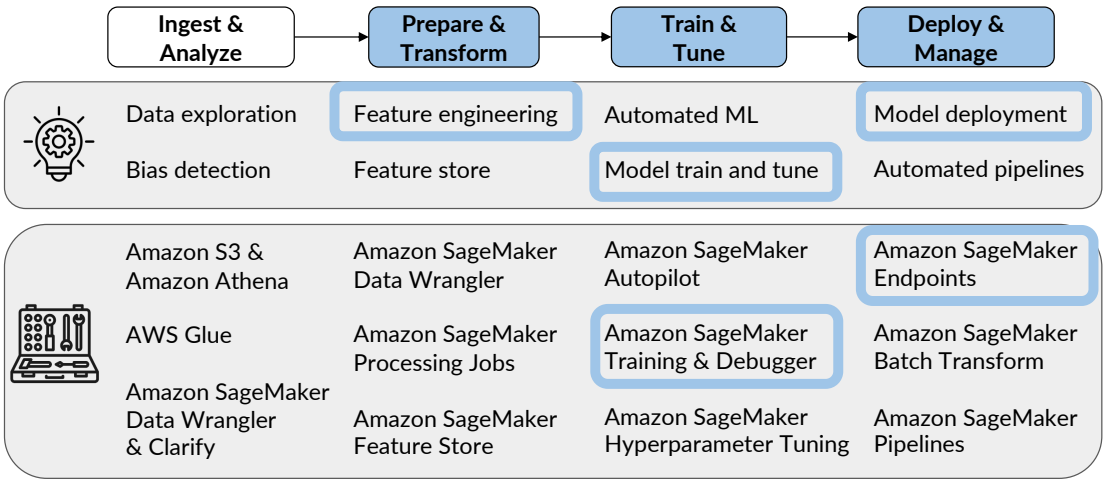
DeepLearning.AI

aws

Practical Data Science

Built-In Algorithms

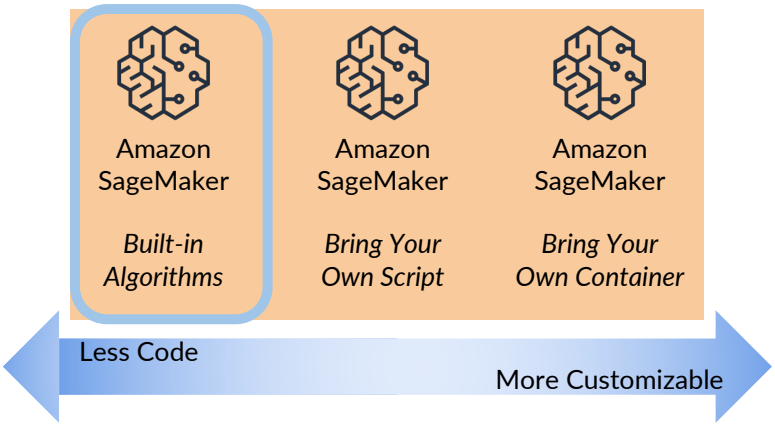
Machine Learning Workflow



Why use built-in algorithms?

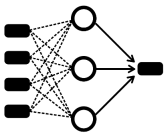
- Implementations are highly-optimized and scalable
- Focus more on domain-specific tasks rather than managing low-level model code and infrastructure
- Trained model can be downloaded and re-used elsewhere

When to choose built-in algorithms vs. custom code



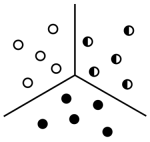
Use cases and algorithms

Popular ML tasks and learning paradigms



Classification & Regression

Supervised



Clustering

Unsupervised

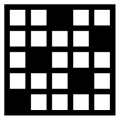


Image Processing

Computer Vision

Text Analysis

NLP / NLU

Classification & regression

Example problems and use cases	Problem types	Input format	Built-in algorithms
Predict if an item belongs to a category: an email spam filter	Binary/multi-class classification	Tabular	XGBoost, K-Nearest Neighbors (k-NN)
Predict a numeric/continuous value: estimate the value of a house	Regression	Tabular	Linear Learner, XGBoost
Predict sales on a new product based on previous sales data	Time-series forecasting	Tabular	DeepAR Forecasting

Clustering

Example problems and use cases	Problem types	Input format	Built-in algorithms
Drop weak features such as the color of a car when predicting its mileage.	Feature engineering: reduce dimensions	Tabular	Principal Component Analysis (PCA)
Detect abnormal behavior	Anomaly detection	Tabular	Random Cut Forest (RCF)
Group high/medium/low-spending customers from transaction histories	Clustering / grouping	Tabular	K-Means
Organize a set of documents into topics based on words and phrases	Topic modeling	Text	Latent Dirichlet Allocation (LDA), Neural Topic Model (NTM)

Image processing

Example problems and use cases	Problem types	Input format	Built-in algorithms
Content moderation	Image classification	Image	Image Classification
Detect people and objects in an image	Object detection	Image	Object Detection
Self-driving cars identify objects in their path	Computer vision	Image	Semantic Segmentation

Text analysis

Example problems and use cases	Problem types	Input format	Built-in algorithms
Convert Spanish to English	Machine translation	Text	Sequence-to-Sequence
Summarize a research paper	Text summarization	Text	Sequence-to-Sequence
Transcribe call center conversations	Speech-to-text	Text	Sequence-to-Sequence
Classify reviews into categories	Text classification	Text	BlazingText

Text analysis

Evolution of text analysis algorithms

Word2Vec
Jan 2013

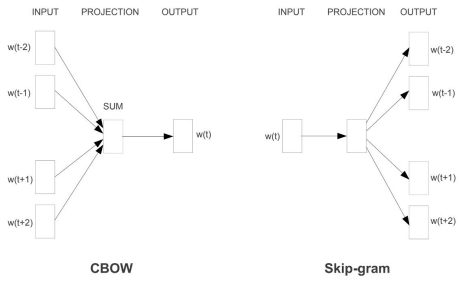
Text analysis algorithm - Word2Vec

Concepts

- Convert text into vectors called “embeddings”
- 300-dimensional vector space
- Perform machine learning on the vectors

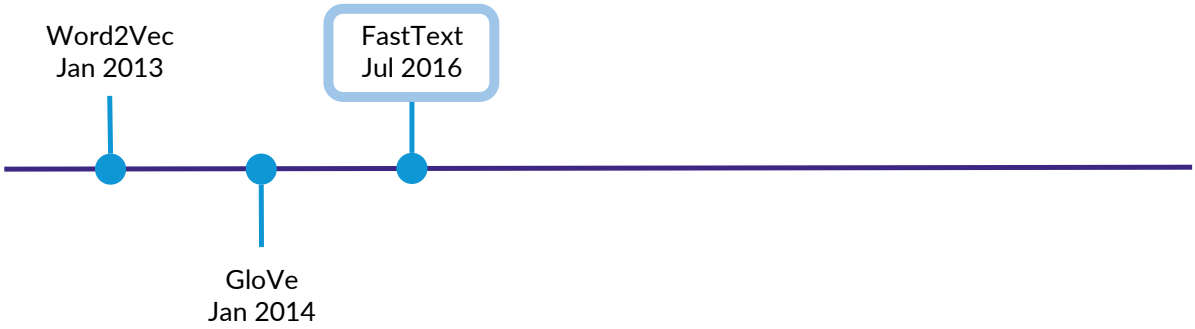
Model architectures to create the embeddings

- Continuous bag-of-words (CBOW)
- Continuous skip-gram



Source: "Efficient Estimation of Word Representations in Vector Space", Mikolov et al., 2013

Evolution of text analysis algorithms



Text analysis algorithm - FastText

Concepts

- Extension of word2vec
- Breaks the word into character sets of length n (n-grams):

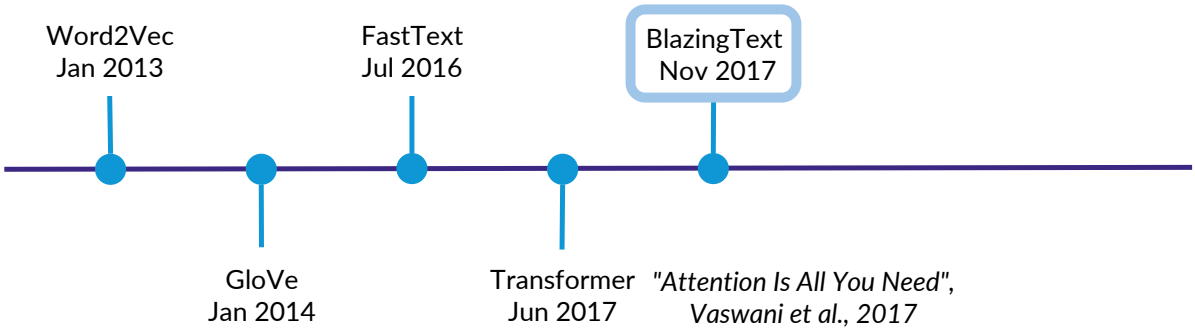
  `"amazon" => "a", "am", "ama", "amaz", "amazo", "amazon"`
- Embedding for a word is the aggregate of the embedding of each n-gram within the word

Implementation

- CBOW and skip-gram models
- Adds text classification

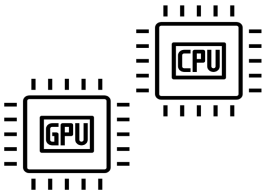
Helps with the out-of-vocabulary (OOV) issue with word2vec

Evolution of text analysis algorithms

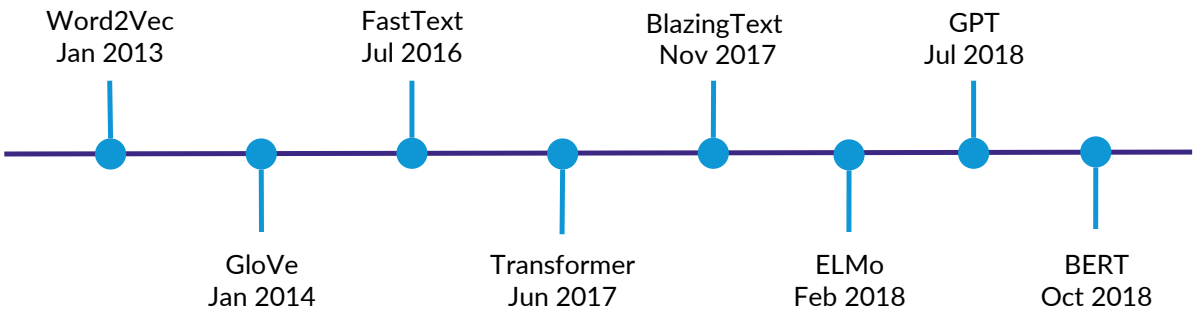


Text analysis algorithm - BlazingText

- Scales and accelerates Word2Vec using multiple CPUs or GPUs for training
- Extends FastText to use GPU acceleration with custom CUDA kernels
- Creates n-gram embeddings using CBOW and skip-gram
- Saves money by early-stopping a training job
  - when the validation accuracy stops increasing
- Optimized I/O for datasets stored in Amazon S3



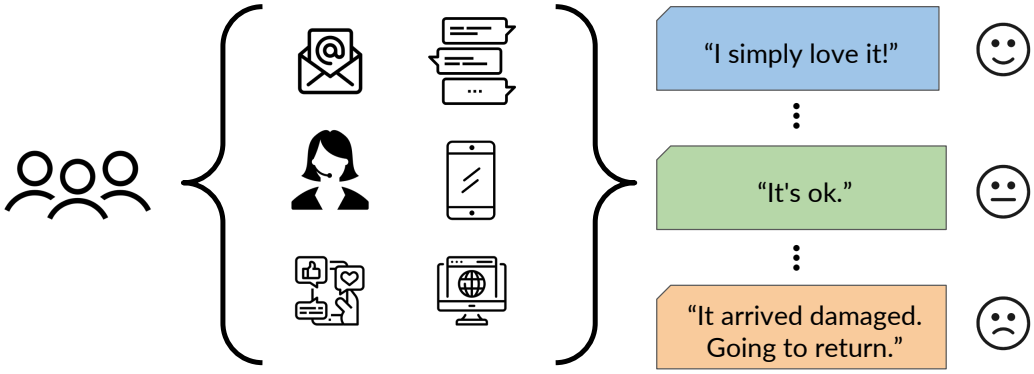
Evolution of text analysis algorithms



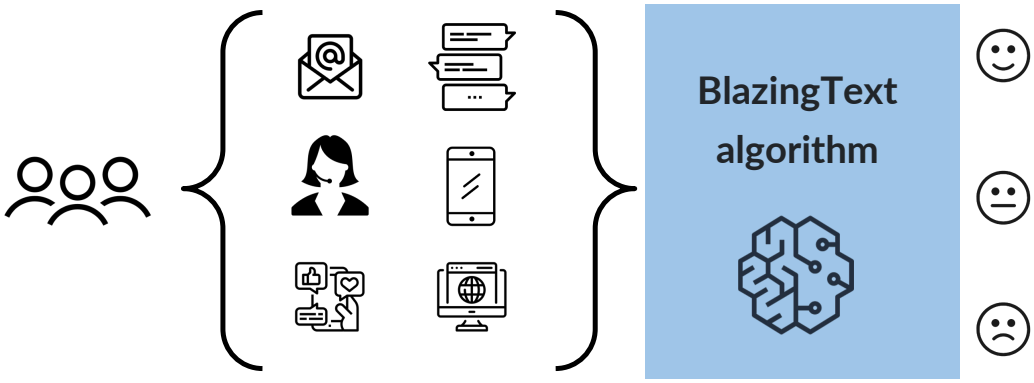
Train a text classifier

with Amazon SageMaker BlazingText

Multi-class classification for sentiment analysis of product reviews



Multi-class classification for sentiment analysis of product reviews



Transform raw review data into features

Ingest & Analyze → **Prepare & Transform** → Train & Tune → Deploy & Manage

```
sentiment,review_body
1,"i simply love it"
0,"it's ok"
-1,"it arrived damaged. going to return"
```

```
__label__1 "i simply love it ."
__label__0 "it's ok ."
__label__-1 "it arrived damaged ."
```


Transform raw review data into features

Ingest & Analyze → **Prepare & Transform** → Train & Tune → Deploy & Manage

NLTK

```
def tokenize(review):
    return nltk.word_tokenize(review)
```

```
__label__1 "i simply love it ."
__label__0 "it's ok ."
__label__-1 "it arrived damaged, going to return ."
```


Amazon SageMaker BlazingText hyper-parameters for text classification

Parameter Name	Recommended Ranges or Values	Description
epochs	[5-15]	Number of complete passes through the dataset
learning_rate	[0.005-0.01]	Step size for the numerical optimizer
min_count	[0-100]	Discard words that appear less than this number
vector_dim	[32-300]	Number of dimensions in vector space
word_ngrams	[1-3]	Number of words n-gram features to use
early_stopping	True or False	Stop training if validation accuracy stops improving
patience	[5-15]	Number of epochs before early stopping

Train a text classifier using Amazon SageMaker BlazingText

Ingest & Analyze → Prepare & Transform → Train & Tune → Deploy & Manage

```
train_data = sagemaker.inputs.TrainingInput(...)
validation_data = sagemaker.inputs.TrainingInput(...)

data_channels = {
    'train': train_data,
    'validation': validation_data
}

image_uri = sagemaker.image_uris.retrieve(framework='blazingtext', ...)

estimator = sagemaker.estimator.Estimator(image_uri=image_uri, ...)
estimator.set_hyperparameters(...)
estimator.fit(...)
```

Retrieves Amazon ECR image URIs for pre-built SageMaker Docker images.

Evaluate the classifier

Ingest & Analyze → Prepare & Transform → Train & Tune → Deploy & Manage

time	metric_name	value
00.0	train:accuracy	0.4865
10.0	train:accuracy	0.5220
20.0	validation:accuracy	0.5364

Deploy the text classifier

and make predictions

Deploy the text classifier

Ingest & Analyze → Prepare & Transform → Train & Tune → **Deploy & Manage**

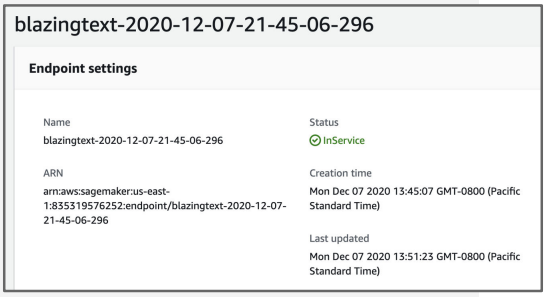
```
text_classifier = estimator.deploy(
    initial_instance_count=1,
    instance_type='ml.m4.xlarge', ...)
```

Increase instance_count > 1 to easily scale out

Deploy the text classifier

Ingest & Analyze → Prepare & Transform → Train & Tune → **Deploy & Manage**

```
text_classifier = estimator.deploy(
    initial_instance_count=1,
    instance_type='ml.m4.xlarge', ...)
```



Deploy the text classifier

Ingest & Analyze → Prepare & Transform → Train & Tune → **Deploy & Manage**

```
text_classifier = estimator.deploy(
    initial_instance_count=1,
    instance_type='ml.m4.xlarge', ...)

payload = {'instances': ['This product is great']}
response = text_classifier.predict(...)

## Sample response:
[{
    "label": ["__label__1"],
    "prob": [0.9506041407585144]
}]
```

Sample prediction request

Prediction response and probability score

Train a text classifier using Amazon SageMaker BlazingText built-in algorithm

Introduction

In this lab you will use SageMaker BlazingText built-in algorithm to predict the sentiment for each customer review. BlazingText is a variant of FastText which is based on word2vec. For more information on BlazingText, see the documentation here: https://docs.aws.amazon.com/sagemaker/latest/dg/blazingtext.html (https://docs.aws.amazon.com/sagemaker/latest/dg/blazingtext.html)

Table of Contents

- 1. Prepare dataset
  - 1.1. Load the dataset
  - 1.2. Transform the dataset
    - Exercise 1
  - 1.3. Split the dataset into train and validation sets
  - 1.4. Upload the `train` and `validation` datasets to S3 bucket
- 2. Train the model
  - Exercise 2
  - Exercise 3
  - Exercise 4
  - Exercise 5
  - Exercise 6
  - Exercise 7
- 3. Deploy the model
- 4. Test the model

Let's install and import required modules.

In [2]:
```
# please ignore warning messages during the installation
!pip install --disable-pip-version-check -q sagemaker==2.35.0
!pip install --disable-pip-version-check -q nltk==3.5
```

```
WARNING: Running pip as the 'root' user can result in broken permi
ssions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead: https://pi
p.pypa.io/warnings/venv (https://pip.pypa.io/warnings/venv)
WARNING: Running pip as the 'root' user can result in broken permi
ssions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead: https://pi
p.pypa.io/warnings/venv (https://pip.pypa.io/warnings/venv)
```

In [3]:
```
import boto3
import sagemaker
import pandas as pd
import numpy as np
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c1/w4')

# 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
```

In [4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

1. Prepare dataset

Let's adapt the dataset into a format that BlazingText understands. The BlazingText format is as follows:

```
__label__<label> "<features>"
```

Here are some examples:

```
__label__-1 "this is bad"
__label__0 "this is ok"
__label__1 "this is great"
```

Sentiment is one of three classes: negative (-1), neutral (0), or positive (1). BlazingText requires that `__label__` is prepended to each sentiment value.

You will tokenize the `review_body` with the Natural Language Toolkit (`nltk`) for the model training. `nltk` documentation can be found here (https://www.nltk.org/). You will also use `nltk` later in this lab to tokenize reviews to use as inputs to the deployed model.

1.1. Load the dataset

Upload the dataset into the Pandas dataframe:

In [5]:
```
!aws s3 cp 's3://dlai-practical-data-science/data/balanced/womens_c
```

```
download: s3://dlai-practical-data-science/data/balanced/womens_cl
othing_ecommerce_reviews_balanced.csv to ./womens_clothing_ecommer
ce_reviews_balanced.csv
```

In [6]:
```
path = './womens_clothing_ecommerce_reviews_balanced.csv'

df = pd.read_csv(path, delimiter=',')
df.head()
```

Out[6]:

	sentiment	review_body	product_category
0	-1	This suit did nothing for me. the top has zero...	Swim
1	-1	Like other reviewers i saw this dress on the ...	Dresses
2	-1	I wish i had read the reviews before purchasin...	Knits
3	-1	I ordered these pants in my usual size (xl) an...	Legwear
4	-1	I noticed this top on one of the sales associa...	Knits

1.2. Transform the dataset

Now you will prepend `__label__` to each sentiment value and tokenize the review body using `nltk` module. Let's import the module and download the tokenizer:

In [7]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[7]:
```
True
```

To split a sentence into tokens you can use `word_tokenize` method. It will separate words, punctuation, and apply some stemming. Have a look at the example:

In [8]:
```
sentence = "I'm not a fan of this product!"

tokens = nltk.word_tokenize(sentence)
print(tokens)
```

```
['I', "'m", 'not', 'a', 'fan', 'of', 'this', 'product', '!']
```

The output of word tokenization can be converted into a string separated by spaces and saved in the dataframe. The transformed sentences are prepared then for better text understending by the model.

Let's define a `prepare_data` function which you will apply later to transform both training and validation datasets.

Exercise 1

Apply the tokenizer to each of the reviews in the `review_body` column of the dataframe `df` .

In [11]:
```
def tokenize(review):
    # delete commas and quotation marks, apply tokenization and joi
    return ' '.join([str(token) for token in nltk.word_tokenize(str

def prepare_data(df):
    df['sentiment'] = df['sentiment'].map(lambda sentiment : '__lab
    ### BEGIN SOLUTION - DO NOT delete this comment for grading pur
    df['review_body'] = df['review_body'].map(lambda review : token
    ### END SOLUTION - DO NOT delete this comment for grading purpo
    return df
```

Test the prepared function and examine the result.

In [12]:
```
# create a sample dataframe
df_example = pd.DataFrame({
    'sentiment':[-1, 0, 1],
    'review_body':[
        "I don't like this product!",
        "this product is ok",
        "I do like this product!"]
})

# test the prepare_data function
print(prepare_data(df_example))

# Expected output:
#      sentiment                    review_body
# 0  __label__-1  i do n't like this product !
# 1   __label__0            this product is ok
# 2   __label__1      i do like this product !
```

```
     sentiment                    review_body
0  __label__-1  i do n't like this product !
1   __label__0            this product is ok
2   __label__1      i do like this product !
```

Apply the `prepare_data` function to the dataset.

In [13]:
```
df_blazingtext = df[['sentiment', 'review_body']].reset_index(drop=
df_blazingtext = prepare_data(df_blazingtext)
df_blazingtext.head()
```

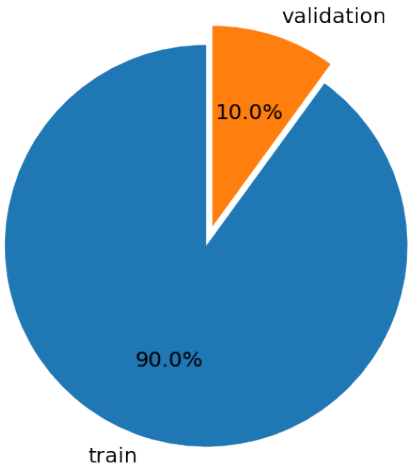
Out[13]:

	sentiment	review_body
0	__label__-1	this suit did nothing for me . the top has zer...
1	__label__-1	like other reviewers i saw this dress on the c...
2	__label__-1	i wish i had read the reviews before purchasin...
3	__label__-1	i ordered these pants in my usual size (xl) ...
4	__label__-1	i noticed this top on one of the sales associa...

1.3. Split the dataset into train and validation sets

Split and visualize a pie chart of the train (90%) and validation (10%) sets. You can do the split using the `sklearn` model function.

In [14]:
```
from sklearn.model_selection import train_test_split

# Split all data into 90% train and 10% holdout
df_train, df_validation = train_test_split(df_blazingtext,
                                           test_size=0.10,
                                           stratify=df_blazingtext[

labels = ['train', 'validation']
sizes = [len(df_train.index), len(df_validation.index)]
explode = (0.1, 0)

fig1, ax1 = plt.subplots()

ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', s

# Equal aspect ratio ensures that pie is drawn as a circle.
ax1.axis('equal')

plt.show()
print(len(df_train))
```




```
6399
```

Save the results as CSV files.

In [15]:
```
blazingtext_train_path = './train.csv'
df_train[['sentiment', 'review_body']].to_csv(blazingtext_train_pat
```

In [16]:
```
blazingtext_validation_path = './validation.csv'
df_validation[['sentiment', 'review_body']].to_csv(blazingtext_vali
```

1.4. Upload the `train` and `validation` datasets to S3 bucket

You will use these to train and validate your model. Let's save them to S3 bucket.

In [17]:
```
train_s3_uri = sess.upload_data(bucket=bucket, key_prefix='blazingt
validation_s3_uri = sess.upload_data(bucket=bucket, key_prefix='bla
```

2. Train the model

Setup the BlazingText estimator. For more information on Estimators, see the SageMaker Python SDK documentation here: https://sagemaker.readthedocs.io/ (https://sagemaker.readthedocs.io/).

Exercise 2

Setup the container image to use for training with the BlazingText algorithm.

Instructions: Use the `sagemaker.image_uris.retrieve` function with the `blazingtext` algorithm.

```
image_uri = sagemaker.image_uris.retrieve(
    region=region,
    framework='...' # the name of framework or algorithm
)
```

In [19]:
```
image_uri = sagemaker.image_uris.retrieve(
    region=region,
    ### BEGIN SOLUTION - DO NOT delete this comment for grading pur
    framework='blazingtext' # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purpo
)
```

Exercise 3

Create an estimator instance passing the container image and other instance parameters.

Instructions: Pass the container image prepared above into the `sagemaker.estimator.Estimator` function.

Note: For the purposes of this lab, you will use a relatively small instance type. Please refer to [this (https://aws.amazon.com/sagemaker/pricing/)](https://aws.amazon.com/sagemaker/pricing/) link for additional instance types that may work for your use case outside of this lab.

In [20]:
```
estimator = sagemaker.estimator.Estimator(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading pur
    image_uri=image_uri, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purpo
    role=role,
    instance_count=1,
    instance_type='ml.m5.large',
    volume_size=30,
    max_run=7200,
    sagemaker_session=sess
)
```

Configure the hyper-parameters for BlazingText. You are using BlazingText for a supervised classification task. For more information on the hyper-parameters, see the documentation here: [https://docs.aws.amazon.com/sagemaker/latest/dg/blazingtext-tuning.html (https://docs.aws.amazon.com/sagemaker/latest/dg/blazingtext-tuning.html)](https://docs.aws.amazon.com/sagemaker/latest/dg/blazingtext-tuning.html)

The hyperparameters that have the greatest impact on word2vec objective metrics are: `learning_rate` and `vector_dim` .

In [21]:
```
estimator.set_hyperparameters(mode='supervised',    # supervised (te
                              epochs=10,            # number of comp
                              learning_rate=0.01,   # step size for
                              min_count=2,          # discard words
                              vector_dim=300,       # number of dime
                              word_ngrams=3)        # number of word
```

To call the `fit` method for the created estimator instance you need to setup the input data channels. This can be organized as a dictionary

```
data_channels = {
    'train': ..., # training data
    'validation': ... # validation data
}
```

where training and validation data are the Amazon SageMaker channels for S3 input data sources.

Exercise 4

Create a train data channel.

Instructions: Pass the S3 input path for training data into the `sagemaker.inputs.TrainingInput` function.

In [22]:
```
train_data = sagemaker.inputs.TrainingInput(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading pur
    train_s3_uri, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purpo
    distribution='FullyReplicated',
    content_type='text/plain',
    s3_data_type='S3Prefix'
)
```

Exercise 5

Create a validation data channel.

Instructions: Pass the S3 input path for validation data into the `sagemaker.inputs.TrainingInput` function.

In [23]:
```
validation_data = sagemaker.inputs.TrainingInput(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading pur
    validation_s3_uri, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purpo
    distribution='FullyReplicated',
    content_type='text/plain',
    s3_data_type='S3Prefix'
)
```

Exercise 6

Organize the data channels defined above as a dictionary.

In [24]:
```
data_channels = {
    ### BEGIN SOLUTION - DO NOT delete this comment for grading pur
    'train': train_data, # Replace None
    'validation': validation_data # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purpo
}
```

Exercise 7

Start fitting the model to the dataset.

Instructions: Call the `fit` method of the estimator passing the configured train and validation inputs (data channels).

```
estimator.fit(
    inputs=..., # train and validation input
    wait=False # do not wait for the job to complete before
continuing
)
```

In [25]:
```
estimator.fit(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading pur
    inputs=data_channels, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purpo
    wait=False
)

training_job_name = estimator.latest_training_job.name
print('Training Job Name:  {}'.format(training_job_name))
```

```
Training Job Name:  blazingtext-2022-08-30-05-30-59-208
```

Review the training job in the console.

Instructions:

- open the link
- notice that you are in the section `Amazon SageMaker -> Training jobs`
- check the name of the training job, its status and other available information

In [26]:
```
from IPython.core.display import display, HTML

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

Review [Training job (https://console.aws.amazon.com/sagemaker/home?region=us-east-1#/jobs/blazingtext-2022-08-30-05-30-59-208)](https://console.aws.amazon.com/sagemaker/home?region=us-east-1#/jobs/blazingtext-2022-08-30-05-30-59-208)

Review the Cloud Watch logs (after about 5 minutes).

Instructions:

- open the link
- open the log stream with the name, which starts from the training job name
- have a quick look at the log messages

In [28]:
```
from IPython.core.display import display, HTML

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

Review [CloudWatch logs (https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#logStream:group=/aws/sagemaker/TrainingJobs;prefix=blazingtext-2022-08-30-05-30-59-208;streamFilter=typeLogStreamPrefix)](https://console.aws.amazon.com/cloudwatch/home?region=us-east-1#logStream:group=/aws/sagemaker/TrainingJobs;prefix=blazingtext-2022-08-30-05-30-59-208;streamFilter=typeLogStreamPrefix) (after about 5 minutes)

Wait for the training job to complete.

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

In [29]:
```
%%time

estimator.latest_training_job.wait(logs=False)
```

```
2022-08-30 05:33:43 Starting - Preparing the instances for trainin
g
2022-08-30 05:33:43 Downloading - Downloading input data
2022-08-30 05:33:43 Training - Training image download completed.
Training in progress.....
2022-08-30 05:34:20 Uploading - Uploading generated training model
.............................................................
.........................
2022-08-30 05:42:06 Completed - Training job completed
CPU times: user 422 ms, sys: 67.4 ms, total: 489 ms
Wall time: 8min 11s
```

Review the train and validation accuracy.

Ignore any warnings.

In [30]:
```
estimator.training_job_analytics.dataframe()
```

```
Warning: No metrics called train:mean_rho found
```

Out[30]:

	timestamp	metric_name	value
0	0.0	train:accuracy	0.5273
1	0.0	validation:accuracy	0.5288

Review the trained model in the S3 bucket.

Instructions:

- open the link
- notice that you are in the section `Amazon S3 -> [bucket name] -> [training job name]` (Example: `Amazon S3 -> sagemaker-us-east-1-82XXXXXXXXXXX -> blazingtext-20XX-XX-XX-XX-XX-XX-XXX`)
- check the existence of the `model.tar.gz` file in the `output` folder

In [31]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="blank" href="https://s3.console.
```

Review [Trained model (https://s3.console.aws.amazon.com/s3/buckets/sagemaker-us-east-1-430732077722/blazingtext-2022-08-30-05-30-59-208/output/?region=us-east-1&tab=overview)](https://s3.console.aws.amazon.com/s3/buckets/sagemaker-us-east-1-430732077722/blazingtext-2022-08-30-05-30-59-208/output/?region=us-east-1&tab=overview) in S3

3. Deploy the model

Now deploy the trained model as an Endpoint.

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

In [32]:
```
%%time

text_classifier = estimator.deploy(initial_instance_count=1,
                                   instance_type='ml.m5.large',
                                   serializer=sagemaker.serializers
                                   deserializer=sagemaker.deseriali

print()
print('Endpoint name:  {}'.format(text_classifier.endpoint_name))
```

```
------!
Endpoint name:  blazingtext-2022-08-30-05-43-04-582
CPU times: user 116 ms, sys: 23 ms, total: 139 ms
Wall time: 3min 2s
```

Review the endpoint in the AWS console.

Instructions:

- open the link
- notice that you are in the section `Amazon SageMaker -> Endpoints -> [Endpoint name]` (Example: `Amazon SageMaker -> Endpoints -> blazingtext-20XX-XX-XX-XX-XX-XX-XXX`)
- check the status and other available information about the Endpoint

In [33]:
```
from IPython.core.display import display, HTML

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

Review [SageMaker REST Endpoint (https://console.aws.amazon.com/sagemaker/home?region=us-east-1#/endpoints/blazingtext-2022-08-30-05-43-04-582)](https://console.aws.amazon.com/sagemaker/home?region=us-east-1#/endpoints/blazingtext-2022-08-30-05-43-04-582)

4. Test the model

Import the `nltk` library to convert the raw reviews into tokens that BlazingText recognizes.

In [34]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[34]:
```
True
```

Specify sample reviews to predict the sentiment.

In [35]:
```
reviews = ['This product is great!',
           'OK, but not great',
           'This is not the right product.']
```

Tokenize the reviews and specify the payload to use when calling the REST API.

In [36]:
```
tokenized_reviews = [' '.join(nltk.word_tokenize(review)) for revie

payload = {"instances" : tokenized_reviews}
print(payload)
```

```
{'instances': ['This product is great !', 'OK , but not great', 'T
his is not the right product .']}
```

Now you can predict the sentiment for each review. Call the `predict` method of the text classifier passing the tokenized sentence instances (`payload`) into the data argument.

In [37]:
```
predictions = text_classifier.predict(data=payload)
for prediction in predictions:
    print('Predicted class: {}'.format(prediction['label'][0].lstri
```

```
Predicted class: 1
Predicted class: -1
Predicted class: -1
```

Upload the notebook into S3 bucket for grading purposes.

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

In [38]:
```
!aws s3 cp ./C1_W4_Assignment.ipynb s3://$bucket/C1_W4_Assignment_L
```

```
upload: ./C1_W4_Assignment.ipynb to s3://sagemaker-us-east-1-43073
2077722/C1_W4_Assignment_Learner.ipynb
```

Please go to the main lab window and click on `Submit` button (see the `Finish the lab` section of the instructions).

In [ ]:
```

```

DeepLearning.AI

aws

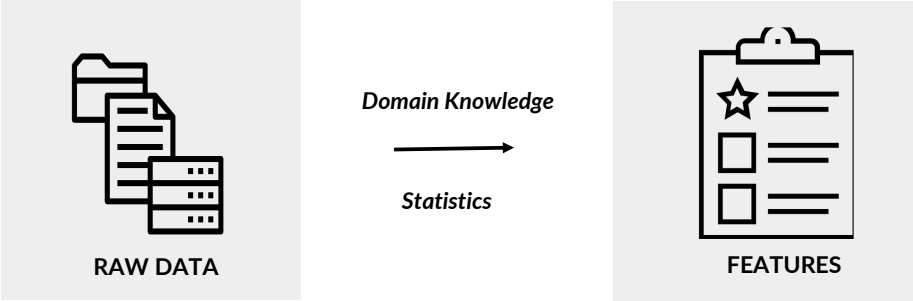
Transform Raw Data into Features for Model Training

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

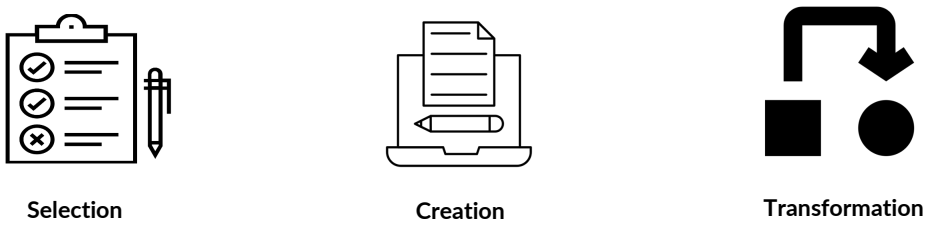
Feature Engineering

Feature Engineering

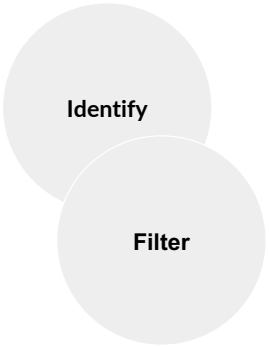


- Dataset best fits the algorithm
- Improve ML model performance

Feature Engineering - Components



Feature Engineering - Selection

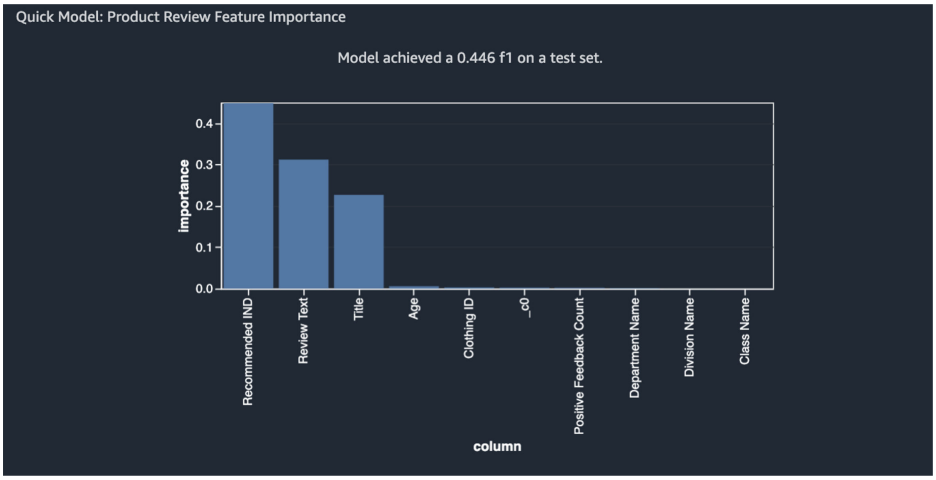


Data attributes

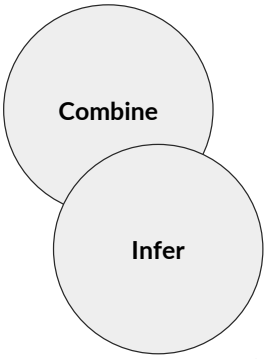
Irrelevant and redundant attributes

- Reduce feature dimensionality
- Train models faster

Feature Importance Report



Feature Engineering - Creation

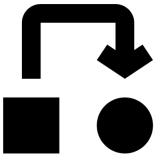


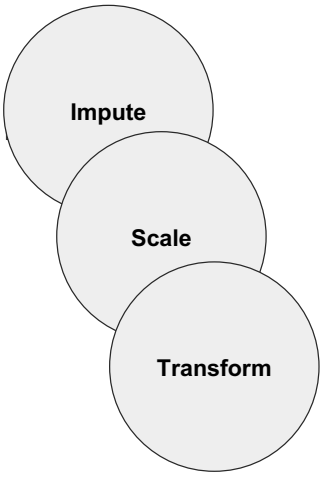
Existing data points into new features

New attributes

✓ Lead to more accurate predictions

Feature Engineering - Transformation





Missing feature values

Numerical features

Non Numerical features

Feature Engineering - Transformation

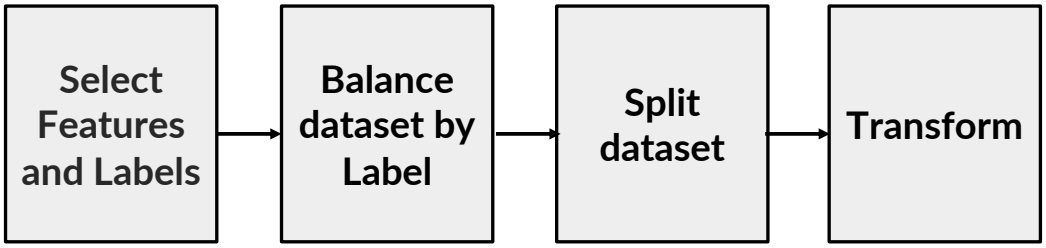
Class Name	Review Text
Blouses	“I simply love it!”
Pants	“It's ok.”
Dresses	“It arrived damaged. Going to return.”

Feature Transformation

Review Text	BERT vecto			
“I simply love it!”	101	2023	...	...
“It's ok.”	3319	1012	...	...
“It arrived damaged. Going to return.”	2003	2307	...	...

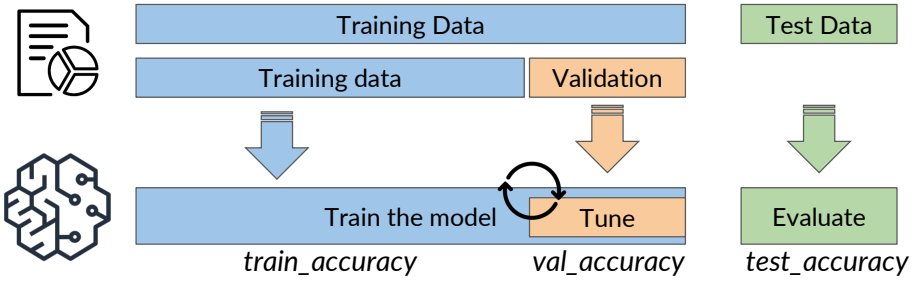
Feature Engineering Pipeline

Feature engineering pipeline

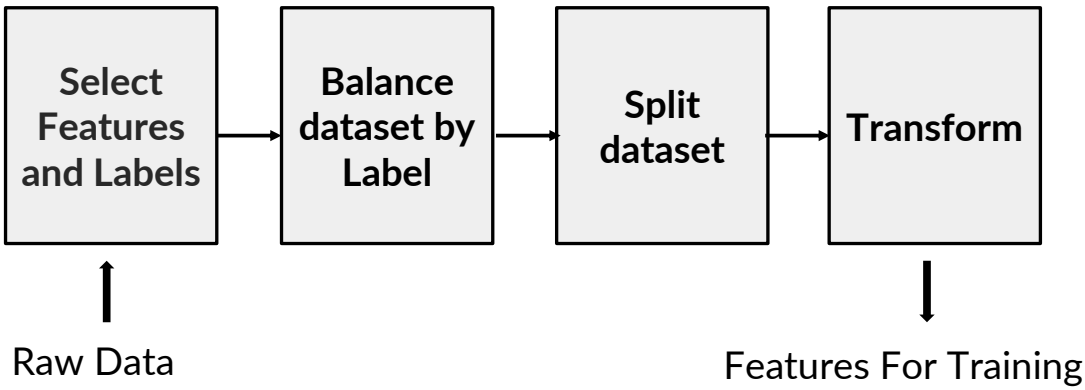


Split Dataset

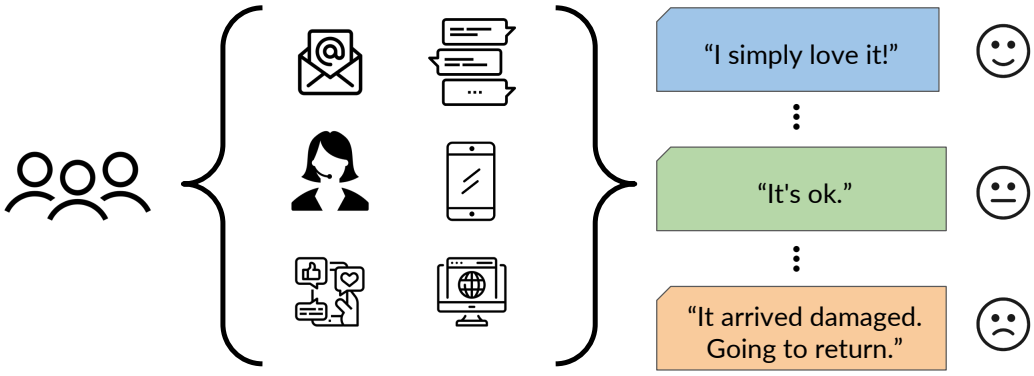
- Training, validation and test data



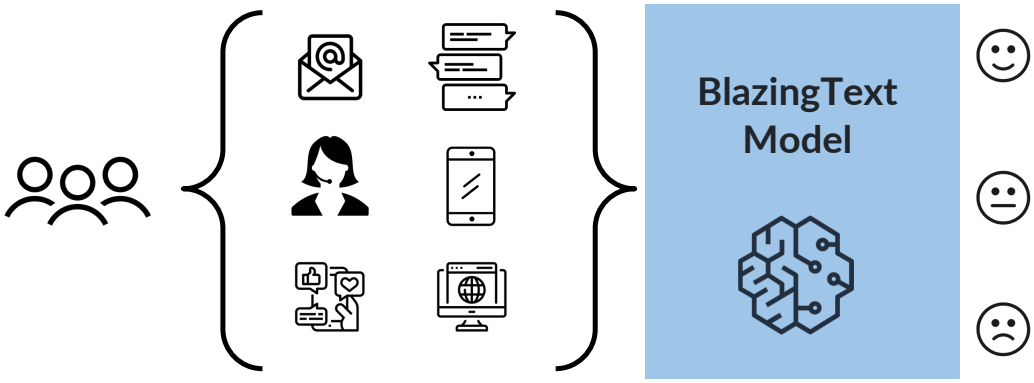
Feature Engineering Pipeline



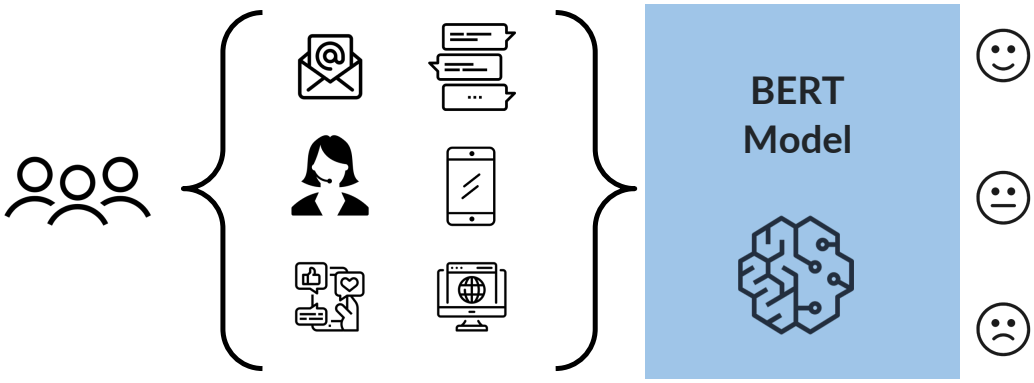
Multi-class Classification for Sentiment Analysis of Product Reviews



Multi-class Classification for Sentiment Analysis of Product Reviews



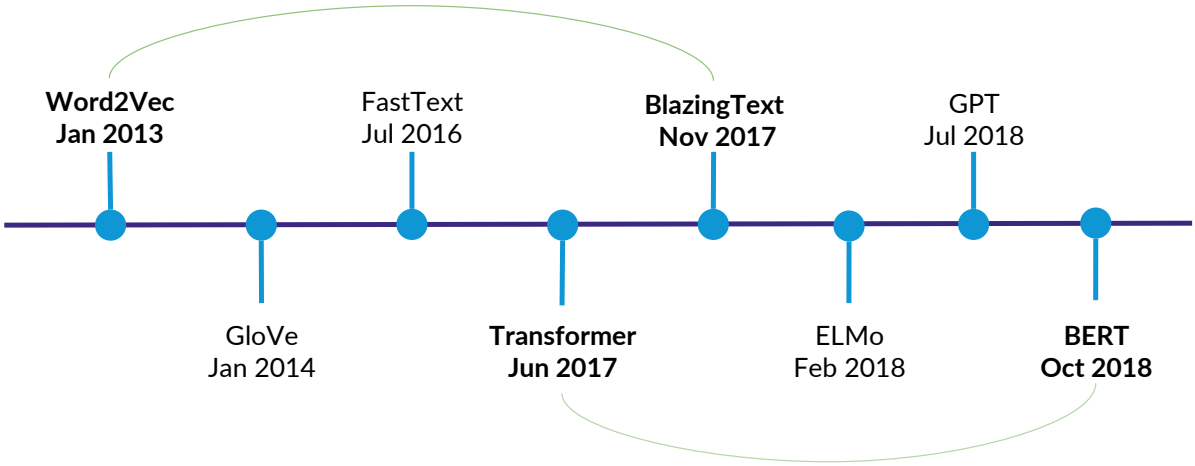
Multi-class Classification for Sentiment Analysis of Product Reviews



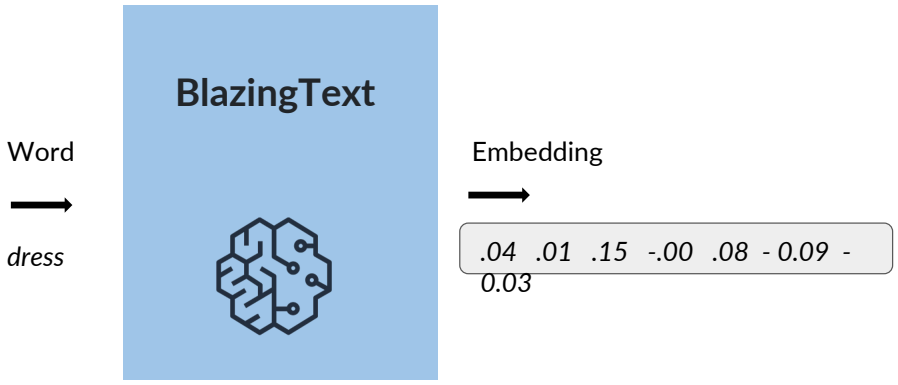
BERT

Bidirectional Encoder
Representations
from Transformers

BlazingText vs BERT



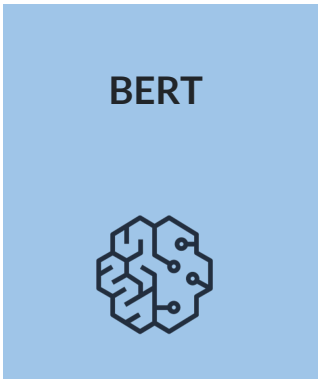
BlazingText - Word Level Embeddings



BERT - Contextual Embeddings

Sentence

I love the dress



Embedding (Token + Segment + Position)

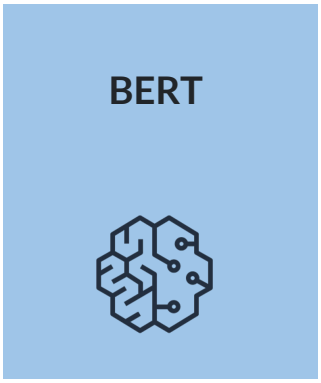
Fixed length embedding

BERT - Contextual Embeddings

Sentence

I love the dress

I love the dress, but not the price



Embedding (Token + Segment + Position)

Fixed length embedding

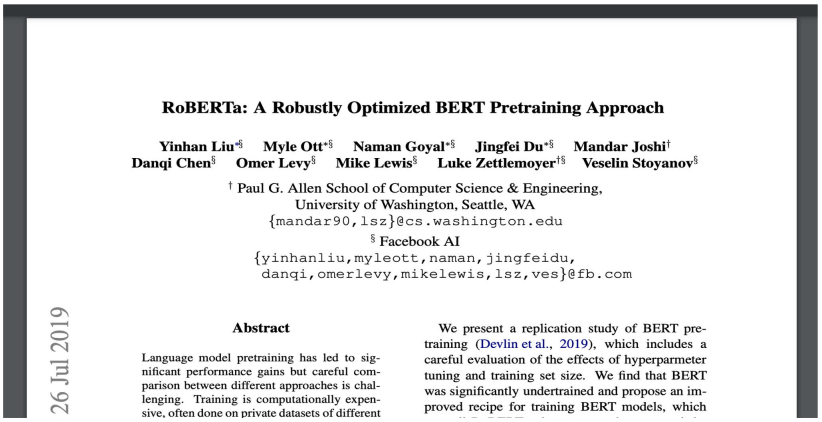
BERT Embeddings

Input for BERT Model	(1, 4, 768)	**Element wise sum of position, segment and token embedding**
POSITION EMBEDDING Input ID +	0 | 1 | 2 | 3 (1, 4, 768)	**Index position in input sequence**
SEGMENT EMBEDDING Segment ID +	0 | 0 | 0 | 0 (1, 4, 768)	**0 = Sentence 1** **1 = Sentence 2**
TOKEN EMBEDDING Input ID +	101 | 2293 | 2023 | 4377 (1, 4, 768)	**Lookup the 768 dimension vector dimension**
Word Piece Tokenization	[CLS], Love, this, dress	**1 input sequence (consisting of 4 tokens)**
Raw Input **sequence**	Love this dress	**1 input**

Feature Engineering

At scale with Amazon SageMaker Processing Jobs

RoBERTa model



RoBERTa: A Robustly Optimized BERT Pretraining Approach

Yinhan Liu[*§] Myle Ott[*§] Naman Goyal[*§] Jingfei Du[*§] Mandar Joshi[†]
Danqi Chen[§] Omer Levy[§] Mike Lewis[§] Luke Zettlemoyer[†§] Veselin Stoyanov[§]

[†] Paul G. Allen School of Computer Science & Engineering,
University of Washington, Seattle, WA
{mandar90,lsz}@cs.washington.edu

[§] Facebook AI
{yinhanliu,myleott,naman,jingfeidu,
danqi,omerlevy,mikelewis,lsz,ves}@fb.com

26 Jul 2019

Abstract

Language model pretraining has led to significant performance gains but careful comparison between different approaches is challenging. Training is computationally expensive, often done on private datasets of different

We present a replication study of BERT pretraining (Devlin et al., 2019), which includes a careful evaluation of the effects of hyperparmeter tuning and training set size. We find that BERT was significantly undertrained and propose an improved recipe for training BERT models, which

BERT Embeddings with RoBERTa

```
from transformers import RobertaTokenizer

PRE_TRAINED_MODEL_NAME = 'roberta-base'



tokenizer =
RobertaTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

Import the Tokenizer class

Create the tokenizer to use based on pre trained model

BERT Embeddings with scikit-learn

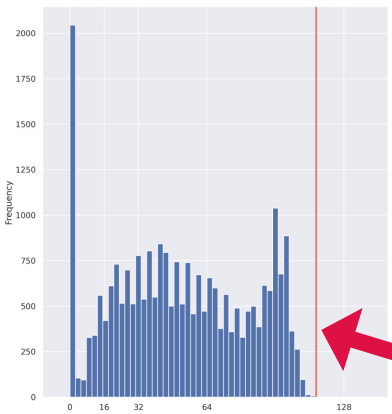
```
def convert_to_bert_input_ids(...):
    encode_plus = tokenizer.encode_plus(
        review,
        add_special_tokens=True,
        max_length=128,
        return_token_type_ids=False,
        padding='max_length',
        return_attention_mask=True,
        return_tensors='pt',
        truncation=True

    return encode_plus['input_ids'].flatten().tolist()
```

Review to be encoded

Max sequence length

Special tokens

BERT hyper-parameter: **max_seq_length**



mean	52.51
std	31.38
min	1.00
10%	10.00
20%	22.00
30%	32.00
40%	41.00
50%	51.00
60%	61.00
70%	73.00
80%	88.00
90%	97.00
100%	**115.00**

BERT Embeddings with scikit-learn

```
def convert_to_bert_input_ids(...):
    encode_plus = tokenizer.encode_plus(
        review,
        add_special_tokens=True,
        max_length=128,
        return_token_type_ids=False,
        padding='max_length',
        return_attention_mask=True,
        return_tensors='pt',
        truncation=True

    return encode_plus['input_ids'].flatten().tolist()
```

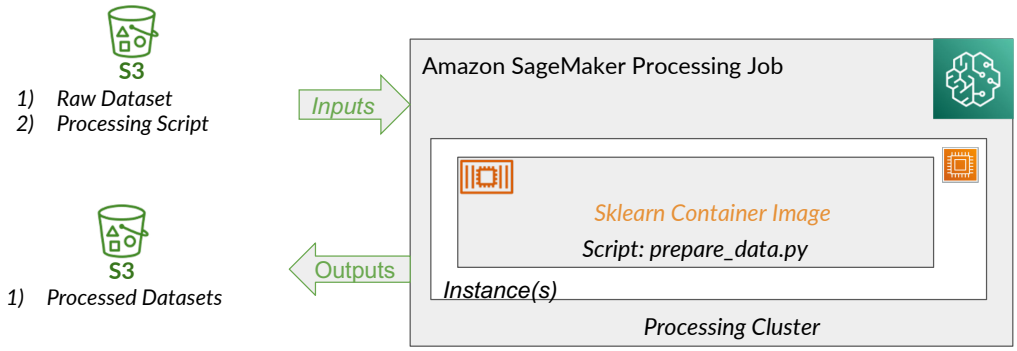
Review to be encoded

Max sequence length

Special tokens

Amazon SageMaker Processing

Execute preprocessing, post processing, model evaluation



Amazon SageMaker Processing with scikit-learn

```
from sagemaker.sklearn.processing import SKLearnProcessor
from sagemaker.processing import ProcessingInput, ProcessingOutput


processor = SKLearnProcessor(
        framework_version='<SCIKIT_LEARN_VERSION',
        role=role,
        instance_type='ml.c5.4xlarge',
        instance_count=2)

processor.run(<parameters>)
```

Setup processing cluster

Run the processing job

Amazon SageMaker Processing with scikit-learn

```
...
code='preprocess-scikit-text-to-bert.py',

inputs=[
    ProcessingInput(
        input_name='raw-input-data',
        source=raw_input_data_s3_uri,
        ...)
],
```

Scikit-learn script to execute

Input data to transform

Amazon SageMaker Processing with scikit-learn

```
...
outputs=[
    ProcessingOutput(
        output_name='bert-train',
        s3_upload_mode='EndOfJob',
        source='/opt/ml/processing/output/bert/train'),
    ...,
],
```

Output from the processing job

Amazon SageMaker Processing with scikit-learn

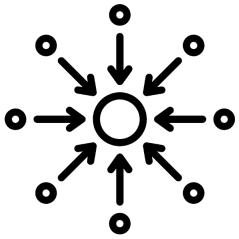
Sentiment	Review
1	*this is a great item!*
-1	*not a good product.*
0	*dress is ok*
-1	*do not use! awful. blah*

SageMaker Processing

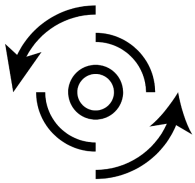
label_id	input_ids		
1	*101*	*2023*	...
-1	*3319*	*1012*	...
0	*2003*	*2307*	...
-1	*102*	*3212*	...

Feature Store

Feature Store



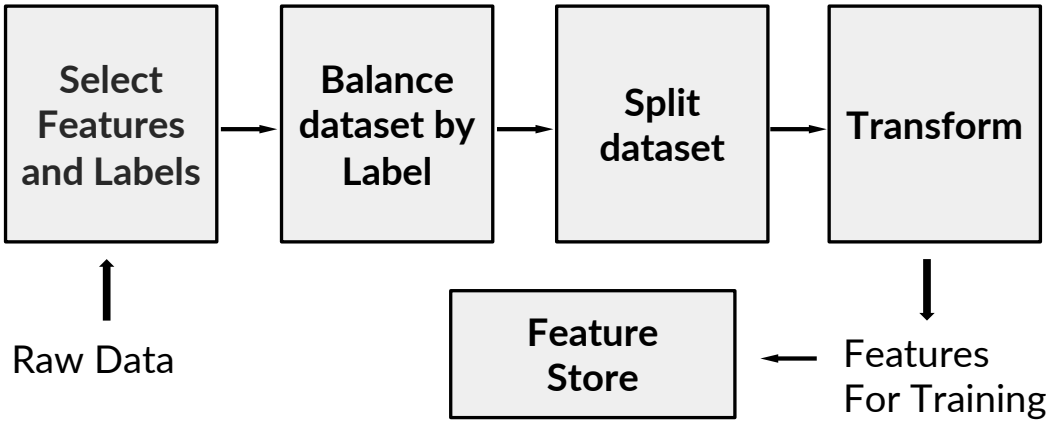
Centralized



Reusable

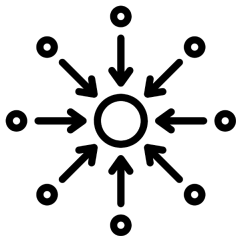
Discoverable

Feature Engineering Pipeline Extended

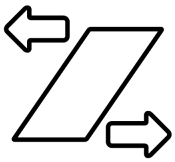


Amazon SageMaker Feature Store

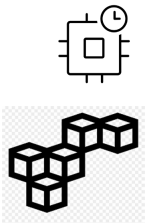
Amazon SageMaker Feature Store



Store and Serve Features



Reduce skew



Real time & Batch

Amazon SageMaker Feature Store - Create

```
from sagemaker.feature_store.feature_group import FeatureGroup

reviews_feature_group_name = "reviews_distilbert_max_seq_length_128"

reviews_feature_group = FeatureGroup(
     name=...,
     feature_definitions=...,
     sagemaker_session=sagemaker_session)

reviews_feature_group.create(
    s3_uri="s3://{}/{}".format(bucket, prefix),
    record_identifier_name=record_identifier_feature_name,
    event_time_feature_name=event_time_feature_name,
    role_arn=role)
```

Name

Create

Amazon SageMaker Feature Store - Ingest

```
reviews_feature_group.ingest(
     data_frame=df_records,
     max_workers=3,
     wait=True)
```

Ingest

Amazon SageMaker Feature Store - Retrieve

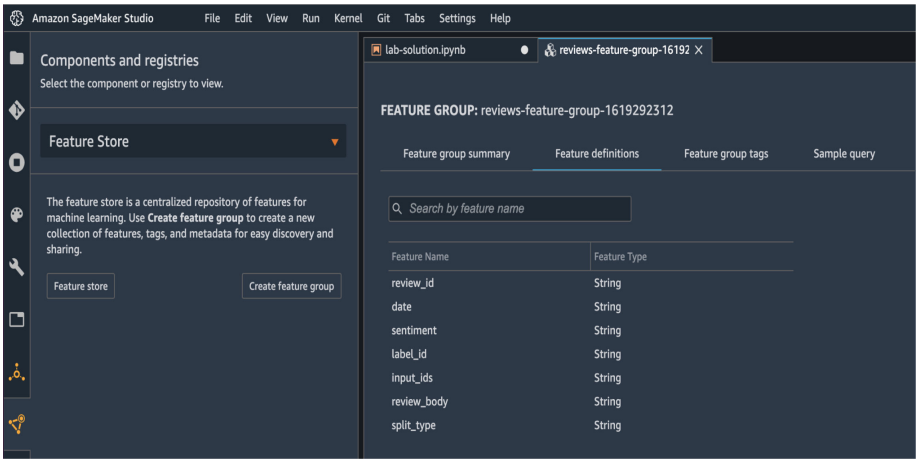
```
reviews_feature_store_query =
    reviews_feature_group.athena_query()

reviews_feature_store_table =
    reviews_feature_store_query.table_name

query_string = 'SELECT review_body, input_ids, input_mask, segment_ids,
label_id FROM "{}" LIMIT 5'.format(reviews_feature_store_table)

reviews_feature_store_query.run(
    query_string=..., ...)
```

Query S3

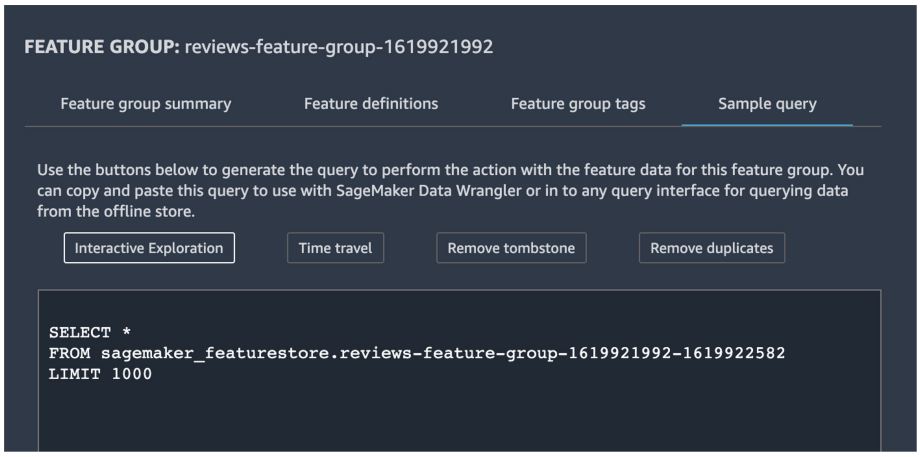
Query string

Execute the query

Amazon SageMaker Feature Store In SageMaker Studio



Amazon SageMaker Feature Store In SageMaker Studio



Amazon SageMaker Feature Store In SageMaker Studio

	date	review_id	sentiment	label_id	input_ids	review_body
0	2021-04-29T18:34:07Z	14136	1	2	[0, 713, 16, 10, 182, 22, 4903, 3760, 254, 22, 2125, 4, 939, 657, 24, 328, 939, 2813, 6215, 74, ...	This is a very "retailer " piece. i love it! i wish retailer would bring back more pieces like t...
1	2021-04-29T18:34:07Z	4026	0	1	[0, 100, 1432, 5, 6173, 8, 2162, 10, 2514, 1836, 11, 5, 2440, 33953, 4, 939, 524, 2333, 10, 1836...	I followed the reviews and bought a larger size in the blue stripe. i am usually a size 8 but or...
2	2021-04-29T18:34:07Z	7522	-1	0	[0, 713, 8443, 16, 98, 11962, 8, 10698, 1969, 8, 939, 657, 5, 32847, 4, 959, 1437, 24, 24232, 90...	This jacket is so cute and fits perfect and i love the motif. however it deposited black linty ...
3	2021-04-29T18:34:07Z	7618	-1	0	[0, 133, 19111, 738, 8, 1421, 738, 32, 2198, 430, 4, 24, 18, 101, 45, 190, 5, 276, 3588, 4, 5, 5...	The catalog shot and model shot are completely different. it's like not even the same dress. the...
4	2021-04-29T18:34:07Z	11942	1	2	[0, 100, 269, 101, 5, 356, 9, 42, 8443, 1437, 53, 939, 206, 24, 1237, 650, 8, 939, 531, 671, 5, ...	I really like the look of this jacket but i think it runs small and i must return the one i rec...

Feature transformation with Amazon SageMaker processing job and Feature Store

Introduction

In this lab you will start with the raw Women's Clothing Reviews dataset and prepare it to train a BERT-based natural language processing (NLP) model. The model will be used to classify customer reviews into positive (1), neutral (0) and negative (-1) sentiment.

You will convert the original review text into machine-readable features used by BERT. To perform the required feature transformation you will configure an Amazon SageMaker processing job, which will be running a custom Python script.

Table of Contents

- 1. Configure the SageMaker Feature Store
  - 1.1. Configure dataset
  - 1.2. Configure the SageMaker feature store
    - Exercise 1
- 2. Transform the dataset
  - Exercise 2
  - Exercise 3
- 3. Query the Feature Store
  - 3.1. Export training, validation, and test datasets from the Feature Store
    - Exercise 4
  - 3.2. Export TSV from Feature Store
  - 3.3. Check that the dataset in the Feature Store is balanced by sentiment
    - Exercise 5
    - Exercise 6
    - Exercise 7

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
```

```
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

## Package Plan ##

  environment location: /opt/conda

  added / updated specs:
    - pytorch==1.6.0


The following packages will be UPDATED:

  ca-certificates    conda-forge::ca-certificates-2022.6.1~ --> pkgs/main
::ca-certificates-2022.07.19-h06a4308_0

The following packages will be SUPERSEDED by a higher-priority channel:

  conda              conda-forge::conda-4.14.0-py37h89c186~ --> pkgs/main
::conda-4.14.0-py37h06a4308_0


Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
Retrieving notices: ...working... done
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]:

```python
import boto3
import sagemaker
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c2/w1')

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

featurestore_runtime = boto3.client(service_name='sagemaker-featurestore-
                                    config=config)

sess = sagemaker.Session(sagemaker_client=sm,
                         sagemaker_featurestore_runtime_client=featuresto

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

1. Configure the SageMaker Feature Store

1.1. Configure dataset

The raw dataset is in the public S3 bucket. Let's start by specifying the S3 location of it:

In [3]:
```
raw_input_data_s3_uri = 's3://dlai-practical-data-science/data/raw/'
print(raw_input_data_s3_uri)
```

```
s3://dlai-practical-data-science/data/raw/
```

List the files in the S3 bucket (in this case it will be just one file):

In [4]:
```
!aws s3 ls $raw_input_data_s3_uri
```

```
2021-04-30 02:21:06    8457214 womens_clothing_ecommerce_reviews.csv
```

1.2. Configure the SageMaker feature store

As the result of the transformation, in addition to generating files in S3 bucket, you will also save the transformed data in the **Amazon SageMaker Feature Store** to be used by others in your organization, for example.

To configure a Feature Store you need to setup a **Feature Group**. This is the main resource containing all of the metadata related to the data stored in the Feature Store. A Feature Group should contain a list of **Feature Definitions**. A Feature Definition consists of a name and the data type. The Feature Group also contains an online store configuration and an offline store configuration controlling where the data is stored. Enabling the online store allows quick access to the latest value for a record via the GetRecord API. The offline store allows storage of the data in your S3 bucket. You will be using the offline store in this lab.

Let's setup the Feature Group name and the Feature Store offline prefix in S3 bucket (you will use those later in the lab):

In [5]:
```
import time
timestamp = int(time.time())

feature_group_name = 'reviews-feature-group-' + str(timestamp)
feature_store_offline_prefix = 'reviews-feature-store-' + str(timestamp)

print('Feature group name: {}'.format(feature_group_name))
print('Feature store offline prefix in S3: {}'.format(feature_store_offli
```

```
Feature group name: reviews-feature-group-1661803330
Feature store offline prefix in S3: reviews-feature-store-1661803330
```

Taking two features from the original raw dataset (`Review Text` and `Rating`), you will transform it preparing to be used for the model training and then to be saved in the Feature Store. Here you will define the related features to be stored as a list of `FeatureDefinition` .

In [6]:
```
from sagemaker.feature_store.feature_definition import (
    FeatureDefinition,
    FeatureTypeEnum,
)

feature_definitions= [
    # unique ID of the review
    FeatureDefinition(feature_name='review_id', feature_type=FeatureTypeE
    # ingestion timestamp
    FeatureDefinition(feature_name='date', feature_type=FeatureTypeEnum.S
    # sentiment: -1 (negative), 0 (neutral) or 1 (positive). It will be f
    FeatureDefinition(feature_name='sentiment', feature_type=FeatureTypeE
    # label ID of the target class (sentiment)
    FeatureDefinition(feature_name='label_id', feature_type=FeatureTypeEn
    # reviews encoded with the BERT tokenizer
    FeatureDefinition(feature_name='input_ids', feature_type=FeatureTypeE
    # original Review Text
    FeatureDefinition(feature_name='review_body', feature_type=FeatureTyp
    # train/validation/test label
    FeatureDefinition(feature_name='split_type', feature_type=FeatureType
]
```

Exercise 1

Create the feature group using the feature definitions defined above.

Instructions: Use the `FeatureGroup` function passing the defined above feature group name and the feature definitions.

```
feature_group = FeatureGroup(
    name=..., # Feature Group name
    feature_definitions=..., # a list of Feature Definitions
    sagemaker_session=sess # SageMaker session
)
```

In [7]:
```
from sagemaker.feature_store.feature_group import FeatureGroup

feature_group = FeatureGroup(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    name=feature_group_name, # Replace None
    feature_definitions=feature_definitions, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    sagemaker_session=sess
)

print(feature_group)
```

```
FeatureGroup(name='reviews-feature-group-1661803330', sagemaker_session=<
sagemaker.session.Session object at 0x7fd212934e10>, feature_definitions=
[FeatureDefinition(feature_name='review_id', feature_type=<FeatureTypeEnu
m.STRING: 'String'>), FeatureDefinition(feature_name='date', feature_type
=<FeatureTypeEnum.STRING: 'String'>), FeatureDefinition(feature_name='sen
timent', feature_type=<FeatureTypeEnum.STRING: 'String'>), FeatureDefinit
ion(feature_name='label_id', feature_type=<FeatureTypeEnum.STRING: 'Strin
g'>), FeatureDefinition(feature_name='input_ids', feature_type=<FeatureTy
peEnum.STRING: 'String'>), FeatureDefinition(feature_name='review_body',
feature_type=<FeatureTypeEnum.STRING: 'String'>), FeatureDefinition(featu
re_name='split_type', feature_type=<FeatureTypeEnum.STRING: 'String'>)])
```

You will use the defined Feature Group later in this lab, the actual creation of the Feature Group will take place in the processing job. Now let's move into the setup of the processing job to transform the dataset.

2. Transform the dataset

You will configure a SageMaker processing job to run a custom Python script to balance and transform the raw data into a format used by BERT model.

Set the transformation parameters including the instance type, instance count, and train/validation/test split percentages. 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 case outside of this lab.

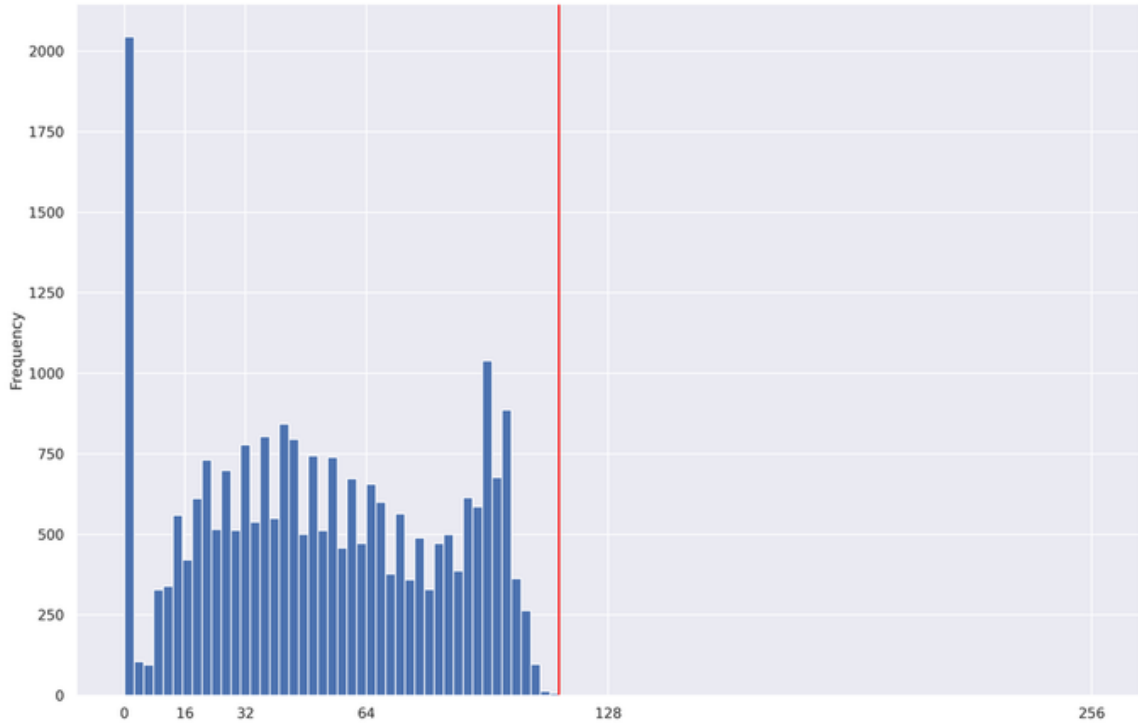
You can also choose whether you want to balance the dataset or not. In this case, you will balance the dataset to avoid class imbalance in the target variable, `sentiment`.

Another important parameter of the model is the `max_seq_length`, which specifies the maximum length of the classified reviews for the RoBERTa model. If the sentence is shorter than the maximum length parameter, it will be padded. In another case, when the sentence is longer, it will be truncated from the right side.

Since a smaller `max_seq_length` leads to faster training and lower resource utilization, you want to find the smallest power-of-2 that captures `100%` of our reviews. For this dataset, the `100th` percentile is `115`. However, it's best to stick with powers-of-2 when using BERT. So let's choose `128` as this is the smallest power-of-2 greater than `115`. You will see below how the shorter sentences will be padded to a maximum length.

```
mean        52.512374
std         31.387048
min          1.000000
10%         10.000000
20%         22.000000
30%         32.000000
```

```
40%          41.000000
50%          51.000000
60%          61.000000
70%          73.000000
80%          88.000000
90%          97.000000
100%        115.000000
max         115.000000
```



In [8]:
```
processing_instance_type='ml.c5.xlarge'
processing_instance_count=1
train_split_percentage=0.90
validation_split_percentage=0.05
test_split_percentage=0.05
balance_dataset=True
max_seq_length=128
```

To balance and transform our data, 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 [9]:
```
from sagemaker.sklearn.processing import SKLearnProcessor

processor = SKLearnProcessor(
    framework_version='0.23-1',
    role=role,
    instance_type=processing_instance_type,
    instance_count=processing_instance_count,
    env={'AWS_DEFAULT_REGION': region},
    max_runtime_in_seconds=7200
)
```

The processing job will be running the Python code from the file `src/prepare_data.py`. In the following exercise you will review the contents of the file and familiarize yourself with main parts of it.

Exercise 2

1. Open the file src/prepare_data.py. Go through the comments to understand its content.
2. Find and review the `convert_to_bert_input_ids()` function, which contains the RoBERTa `tokenizer` configuration.
3. Complete method `encode_plus` of the RoBERTa `tokenizer`. Pass the `max_seq_length` as a value for the argument `max_length`. It defines a pad to a maximum length specified.
4. Save the file src/prepare_data.py (with the menu command File -> Save Python File).

This cell will take approximately 1-2 minutes to run.

In [10]:
```
import sys, importlib
sys.path.append('src/')

# import the `prepare_data.py` module
import prepare_data

# reload the module if it has been previously loaded
if 'prepare_data' in sys.modules:
    importlib.reload(prepare_data)

input_ids = prepare_data.convert_to_bert_input_ids("this product is great

updated_correctly = False

if len(input_ids) != max_seq_length:
    print('##################################################################
    print('Please check that the function \'convert_to_bert_input_ids\' i
    print('##################################################################
    raise Exception('Please check that the function \'convert_to_bert_inp
else:
    print('##################')
    print('Updated correctly!')
    print('##################')

    updated_correctly = True
```

```
##################
Updated correctly!
##################
```

Review the results of tokenization for the given example (|*"this product is great!"*|):

In [11]:
```
input_ids = prepare_data.convert_to_bert_input_ids("this product is great

print(input_ids)
print('Length of the sequence: {}'.format(len(input_ids)))
```

```
[0, 9226, 1152, 16, 372, 328, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Length of the sequence: 128
```

Launch the processing job with the custom script passing defined above parameters.

In [12]:
```
from sagemaker.processing import ProcessingInput, ProcessingOutput

if (updated_correctly):

    processor.run(code='src/prepare_data.py',
                  inputs=[
                        ProcessingInput(source=raw_input_data_s3_uri,
                                        destination='/opt/ml/processing/input
                                        s3_data_distribution_type='ShardedByS
                  ],
                  outputs=[
                        ProcessingOutput(output_name='sentiment-train',
                                         source='/opt/ml/processing/output/se
                                         s3_upload_mode='EndOfJob'),
                        ProcessingOutput(output_name='sentiment-validation',
                                         source='/opt/ml/processing/output/se
                                         s3_upload_mode='EndOfJob'),
                        ProcessingOutput(output_name='sentiment-test',
                                         source='/opt/ml/processing/output/se
                                         s3_upload_mode='EndOfJob')
                  ],
                  arguments=['--train-split-percentage', str(train_split_perc
                             '--validation-split-percentage', str(validation_
                             '--test-split-percentage', str(test_split_percen
                             '--balance-dataset', str(balance_dataset),
                             '--max-seq-length', str(max_seq_length),
                             '--feature-store-offline-prefix', str(feature_st
                             '--feature-group-name', str(feature_group_name)
                  ],
                  logs=True,
                  wait=False)

else:
    print('#######################################')
    print('Please update the code correctly above.')
    print('#######################################')
```

```
Job Name:  sagemaker-scikit-learn-2022-08-29-20-16-18-304
Inputs:  [{'InputName': 'input-1', 'AppManaged': False, 'S3Input': {'S3Ur
i': 's3://dlai-practical-data-science/data/raw/', 'LocalPath': '/opt/ml/p
rocessing/input/data/', 'S3DataType': 'S3Prefix', 'S3InputMode': 'File',
'S3DataDistributionType': 'ShardedByS3Key', 'S3CompressionType': 'None'}}
, {'InputName': 'code', 'AppManaged': False, 'S3Input': {'S3Uri': 's3://s
agemaker-us-east-1-504458356961/sagemaker-scikit-learn-2022-08-29-20-16-1
8-304/input/code/prepare_data.py', 'LocalPath': '/opt/ml/processing/input
/code', 'S3DataType': 'S3Prefix', 'S3InputMode': 'File', 'S3DataDistribut
ionType': 'FullyReplicated', 'S3CompressionType': 'None'}}]
Outputs:  [{'OutputName': 'sentiment-train', 'AppManaged': False, 'S3Outp
ut': {'S3Uri': 's3://sagemaker-us-east-1-504458356961/sagemaker-scikit-le
arn-2022-08-29-20-16-18-304/output/sentiment-train', 'LocalPath': '/opt/m
l/processing/output/sentiment/train', 'S3UploadMode': 'EndOfJob'}}, {'Out
putName': 'sentiment-validation', 'AppManaged': False, 'S3Output': {'S3Ur
i': 's3://sagemaker-us-east-1-504458356961/sagemaker-scikit-learn-2022-08
-29-20-16-18-304/output/sentiment-validation', 'LocalPath': '/opt/ml/proc
essing/output/sentiment/validation', 'S3UploadMode': 'EndOfJob'}}, {'Outp
utName': 'sentiment-test', 'AppManaged': False, 'S3Output': {'S3Uri': 's3
://sagemaker-us-east-1-504458356961/sagemaker-scikit-learn-2022-08-29-20-
16-18-304/output/sentiment-test', 'LocalPath': '/opt/ml/processing/output
/sentiment/test', '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 [13]:
```
scikit_processing_job_name = processor.jobs[-1].describe()['ProcessingJob

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

```
Processing job name: sagemaker-scikit-learn-2022-08-29-20-16-18-304
```

Exercise 3

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 [14]:
```
print(processor.jobs[-1].describe().keys())
```

```
dict_keys(['ProcessingInputs', 'ProcessingOutputConfig', 'ProcessingJobNa
me', 'ProcessingResources', 'StoppingCondition', 'AppSpecification', 'Env
ironment', 'RoleArn', 'ProcessingJobArn', 'ProcessingJobStatus', 'LastMod
ifiedTime', 'CreationTime', 'ResponseMetadata'])
```

In [16]:
```
### 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 [17]:
```
from IPython.core.display import display, HTML

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

Review processing job

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

In [18]:
```
from IPython.core.display import display, HTML

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

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 [19]:
```
from IPython.core.display import display, HTML

display(HTML('<b>Review <a target="blank" href="https://s3.console.aws.am
```

Review S3 output data after the processing job has completed

Wait for the processing job to complete.

This cell will take approximately 15 minutes to run.

In [20]:
```
%%time

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

running_processor.wait(logs=False)
```

```
.............................................................!CPU times: u
ser 250 ms, sys: 42.2 ms, total: 292 ms
Wall time: 5min 5s
```

Please wait until ^^ Processing Job ^^ completes above

Inspect the transformed and balanced data in the S3 bucket.

In [21]:
```
processing_job_description = running_processor.describe()

output_config = processing_job_description['ProcessingOutputConfig']
for output in output_config['Outputs']:
    if output['OutputName'] == 'sentiment-train':
        processed_train_data_s3_uri = output['S3Output']['S3Uri']
    if output['OutputName'] == 'sentiment-validation':
        processed_validation_data_s3_uri = output['S3Output']['S3Uri']
    if output['OutputName'] == 'sentiment-test':
        processed_test_data_s3_uri = output['S3Output']['S3Uri']

print(processed_train_data_s3_uri)
print(processed_validation_data_s3_uri)
print(processed_test_data_s3_uri)
```

```
s3://sagemaker-us-east-1-504458356961/sagemaker-scikit-learn-2022-08-29-2
0-16-18-304/output/sentiment-train
s3://sagemaker-us-east-1-504458356961/sagemaker-scikit-learn-2022-08-29-2
0-16-18-304/output/sentiment-validation
s3://sagemaker-us-east-1-504458356961/sagemaker-scikit-learn-2022-08-29-2
0-16-18-304/output/sentiment-test
```

In [22]:
```
!aws s3 ls $processed_train_data_s3_uri/
```

```
2022-08-29 20:30:16    4890076 part-algo-1-womens_clothing_ecommerce_revi
ews.tsv
```

In [23]:
```
!aws s3 ls $processed_validation_data_s3_uri/
```

```
2022-08-29 20:30:17     264236 part-algo-1-womens_clothing_ecommerce_revi
ews.tsv
```

In [24]:
```
!aws s3 ls $processed_test_data_s3_uri/
```

```
2022-08-29 20:30:17     273558 part-algo-1-womens_clothing_ecommerce_revi
ews.tsv
```

Copy the data into the folder `balanced`.

In [25]:
```
!aws s3 cp $processed_train_data_s3_uri/part-algo-1-womens_clothing_ecomm
!aws s3 cp $processed_validation_data_s3_uri/part-algo-1-womens_clothing_
!aws s3 cp $processed_test_data_s3_uri/part-algo-1-womens_clothing_ecomme
```

```
download: s3://sagemaker-us-east-1-504458356961/sagemaker-scikit-learn-20
22-08-29-20-16-18-304/output/sentiment-train/part-algo-1-womens_clothing_
ecommerce_reviews.tsv to balanced/sentiment-train/part-algo-1-womens_clot
hing_ecommerce_reviews.tsv
download: s3://sagemaker-us-east-1-504458356961/sagemaker-scikit-learn-20
22-08-29-20-16-18-304/output/sentiment-validation/part-algo-1-womens_clot
hing_ecommerce_reviews.tsv to balanced/sentiment-validation/part-algo-1-w
omens_clothing_ecommerce_reviews.tsv
download: s3://sagemaker-us-east-1-504458356961/sagemaker-scikit-learn-20
22-08-29-20-16-18-304/output/sentiment-test/part-algo-1-womens_clothing_e
commerce_reviews.tsv to balanced/sentiment-test/part-algo-1-womens_clothi
ng_ecommerce_reviews.tsv
```

Review the training, validation and test data outputs:

In [26]:
```
!head -n 5 ./balanced/sentiment-train/part-algo-1-womens_clothing_ecommer
```

```
review_id       sentiment       label_id        input_ids
review_body     date
3343    0       1       [0, 133, 1521, 1415, 2422, 11962, 804, 1437, 53,
11, 621, 1437, 5, 1468, 2551, 6162, 4, 888, 1437, 23, 513, 11, 5, 2272, 6
5, 1437, 5, 1468, 16, 430, 11, 5, 29874, 8, 2705, 9042, 131, 114, 24, 56,
70, 57, 5, 29874, 1468, 1437, 24, 74, 33, 57, 203, 35735, 4, 2, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
The design looked super cute online  but in person  the material seemed c
heap. actually  at least in the green one  the material is different in t
he striped and solid sections; if it had all been the striped material  i
t would have been much nicer.   2022-08-29T20:24:48Z
1154    1       2       [0, 100, 2740, 209, 11, 127, 6097, 1836, 1437, 97
3, 1437, 8, 51, 2564, 6128, 6683, 4682, 5, 17507, 4, 13, 162, 1437, 939,
303, 106, 7, 28, 350, 3229, 11, 14, 443, 4, 939, 439, 7, 5, 1400, 8, 1381
, 15, 5, 974, 8, 51, 58, 350, 380, 11, 5, 13977, 4, 939, 109, 5940, 106,
600, 4, 51, 32, 157, 156, 1437, 939, 2638, 5, 10397, 8, 5, 5933, 21, 1969
, 4, 145, 195, 108, 246, 113, 51, 376, 7, 5, 795, 7451, 8, 14, 21, 2051,
19, 162, 4, 492, 106, 10, 860, 1437, 5952, 51, 173, 357, 13, 47, 4, 2, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
"I ordered these in my typical size  26  and they fit everywhere perfectl
y except the butt. for me  i found them to be too tight in that area. i w
ent to the store and tried on the 27 and they were too big in the waist.
i do recommend them though. they are well made  i loved the wash and the
length was perfect. being 5'3"" they came to the lower ankle and that was
fine with me. give them a try  hopefully they work better for you."
2022-08-29T20:24:48Z
19353   -1      0       [0, 100, 21, 182, 6649, 7, 1325, 42, 3588, 19, 10
, 36069, 43805, 1437, 1326, 101, 22196, 219, 1104, 18309, 479, 192, 7391,
3493, 4, 5, 3588, 10199, 16, 182, 2579, 8, 939, 222, 860, 24, 15, 8, 24,
16, 10, 182, 2579, 2564, 1437, 9321, 19, 10, 410, 4140, 53, 127, 821, 521
2, 99, 10, 6162, 2129, 1318, 43805, 14, 1326, 101, 24, 21, 793, 8, 4924,
7, 16227, 142, 5, 4204, 1667, 33, 10, 1104, 22196, 219, 6572, 328, 6215,
47, 269, 240, 7, 1649, 110, 12952, 137, 3981, 7, 110, 8259, 916, 4, 12056
, 5, 1318, 34, 57, 8620, 10, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1]   I was very shocked to receive this dress
with a rusty zipper  looks like crusty white rust . see attached pictures
. the dress fabric is very nice and i did try it on and it is a very nice
fit  lined with a little stretch but my gosh what a cheap poor quality zi
pper that looks like it was old and exposed to moisture because the metal
parts have a white crusty substance! retailer you really need to check yo
ur merchandise before sending to your loyal customers. lately the quality
has been declining a    2022-08-29T20:24:48Z
1465    0       1       [0, 713, 3588, 16, 182, 11962, 8, 10698, 1341, 15
7, 4, 47, 218, 75, 33, 7, 3568, 10, 11689, 19, 24, 1437, 61, 16, 6344, 14
37, 8, 24, 34, 24897, 12189, 4, 5, 13422, 16, 14, 5, 10199, 15, 5, 299, 4
57, 16, 10, 410, 7735, 4, 24, 16, 182, 2016, 8, 13116, 4, 67, 1437, 5, 10
25, 34, 103, 6664, 11918, 8, 5, 1318, 473, 45, 2045, 7, 28, 5, 275, 4, 93
9, 437, 2396, 24, 600, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1]       This dress is very cute and fits quite we
ll. you don't have to wear a bra with it  which is awesome  and it has de
lightful pockets. the downside is that the fabric on the top half is a li
ttle weird. it is very heavy and stiff. also  the inside has some fraying
and the quality does not seem to be the best. i'm keeping it though.
2022-08-29T20:24:48Z
```

In [27]:
```
!head -n 5 ./balanced/sentiment-validation/part-algo-1-womens_clothing_ec
```

```
review_id       sentiment       label_id        input_ids
review_body     date
15360   -1      0       [0, 1708, 1437, 24, 95, 630, 75, 2450, 62, 4, 78,
1437, 5, 9215, 73, 16526, 606, 23983, 12552, 114, 2468, 2829, 8, 24, 16,
182, 1202, 7, 888, 120, 24, 15, 111, 1169, 8296, 88, 24, 50, 6539, 24, 81
, 110, 471, 4, 8, 1437, 683, 15, 1437, 5, 299, 21, 10, 380, 41783, 8, 89,
21, 350, 203, 1468, 23, 5, 2576, 61, 156, 24, 356, 269, 35156, 4, 9574, 1
437, 939, 524, 2609, 42, 7, 28, 5, 403, 19, 98, 171, 9, 5, 1964, 939, 907
, 31, 6215, 12056, 4, 5, 1468, 16, 269, 11962, 1437, 53, 5, 3588, 965, 75
, 966, 5, 425, 1589, 1351, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1]      But  it just doesn't measure up. first  the slip/lining c
omes unattached if pulled slightly and it is very difficult to actually g
et it on - either stepping into it or pulling it over your head. and  onc
e on  the top was a big snug and there was too much material at the botto
m which made it look really bulky. unfortunately  i am finding this to be
the case with so many of the items i buy from retailer lately. the materi
al is really cute  but the dress isn't worth the price / effort.
2022-08-29T20:24:48Z
7838    1       2       [0, 713, 5780, 16, 98, 3035, 328, 5, 6399, 16, 95
, 10, 410, 7082, 98, 645, 110, 4505, 1836, 8, 24, 40, 2564, 6683, 4, 2, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1]      This print is so cool! the shirt is just a little loose s
o order your usual size and it will fit perfectly.      2022-08-29T20:24:
48Z
19592   -1      0       [0, 100, 437, 4716, 1459, 1437, 8, 939, 1467, 215
9, 42, 14, 24, 74, 28, 10, 380, 23204, 111, 14, 16, 5, 356, 14, 16, 15266
, 15, 5, 1421, 4, 5, 23204, 16, 12272, 219, 1437, 5, 5397, 21, 98, 380, 8
, 35156, 939, 56, 3605, 442, 24, 4477, 235, 4, 24, 1326, 372, 15, 5, 1421
, 259, 1437, 53, 45, 15, 162, 4, 164, 124, 4, 98, 29673, 4567, 4, 5, 3195
, 12339, 16, 1528, 7, 99, 16, 2343, 8, 24, 34, 410, 22, 31670, 113, 50, 1
8459, 420, 5, 10762, 4, 98, 89, 16, 10, 5262, 828, 9, 4617, 4, 1374, 1437
, 45, 10, 372, 23204, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1]        "I'm petite  and i knew buying this that it would be a bi
g sweater - that is the look that is portrayed on the model. the sweater
is scratchy  the neck was so big and bulky i had trouble making it lay ri
ght. it looks great on the model here  but not on me. going back. so bumm
ed. the color gray is true to what is shown and it has little ""holes"" o
r openings across the shoulders. so there is a tiny bit of detail. overal
l  not a great sweater."        2022-08-29T20:24:48Z
9833    0       1       [0, 713, 21, 95, 45, 34203, 13, 162, 5579, 13724,
196, 350, 203, 101, 10, 21592, 3588, 4, 939, 64, 75, 269, 109, 10, 12138,
5357, 13977, 98, 2532, 14, 74, 146, 42, 3588, 55, 16106, 13, 951, 1493, 4
, 939, 21, 6661, 7, 33, 7, 671, 142, 24, 1411, 157, 19, 6255, 1536, 8, 10
317, 328, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1]      This was just not flattering for me--looked too much like a mater
nity dress. i can't really do a belted waist so perhaps that would make t
his dress more versatile for someone else. i was sorry to have to return
because it goes well with sandals and boots!    2022-08-29T20:24:48Z
```

In [28]:
```
!head -n 5 ./balanced/sentiment-test/part-algo-1-womens_clothing_ecommerc
```

review_id	sentiment	label_id	input_ids	review_body	date
13528	-1	0	[0, 133, 23204, 16, 182, 3137, 24382, 8, 1415, 205, 5, 78, 86, 939, 1381, 24, 15, 1437, 959, 5, 1468, 16, 7174, 8, 2829, 192, 149, 4, 67, 1437, 71, 65, 3568, 24, 2198, 13596, 66, 8, 156, 5, 251, 21764, 11789, 7, 3568, 396, 14784, 11, 227, 349, 3568, 4, 71, 5, 371, 86, 939, 5328, 24, 1437, 939, 2967, 10, 739, 4683, 11, 5, 4709, 6195, 405, 1437, 144, 533, 528, 7, 5, 10079, 14784, 4, 939, 74, 45, 5940, 42, 23204, 4, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	The sweater is very comfy and looked good the first time i tried it on however the material is thin and slightly see through. also after one wear it completely stretched out and made the long sleeves awkward to wear without washing in between each wear. after the third time i wore it i discovered a large hole in the armpit most likely due to the excessive washing. i would not recommend this sweater.	2022-08-29T20:24:48Z
1089	1	2	[0, 100, 1467, 939, 770, 42, 299, 5, 1151, 939, 794, 24, 1437, 98, 2500, 127, 2813, 8458, 24, 439, 454, 24, 439, 15, 1392, 328, 24, 10698, 16467, 11, 127, 2340, 1836, 8, 939, 657, 14, 24, 416, 34, 10, 22220, 6013, 299, 4, 939, 120, 33391, 8378, 939, 3568, 24, 12, 206, 1368, 417, 11, 2242, 354, 429, 28, 127, 92, 213, 12, 560, 328, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1

3. Query the Feature Store

In addition to transforming the data and saving in S3 bucket, the processing job populates the feature store with the transformed and balanced data. Let's query this data using Amazon Athena.

3.1. Export training, validation, and test datasets from the Feature Store

Here you will do the export only for the training dataset, as an example.

Use `athena_query()` function to create an Athena query for the defined above Feature Group. Then you can pull the table name of the Amazon Glue Data Catalog table which is auto-generated by Feature Store.

In [29]:
```
feature_store_query = feature_group.athena_query()

feature_store_table = feature_store_query.table_name

query_string = """
    SELECT date,
        review_id,
        sentiment,
        label_id,
        input_ids,
        review_body
    FROM "{}"
    WHERE split_type='train'
    LIMIT 5
""".format(feature_store_table)

print('Glue Catalog table name: {}'.format(feature_store_table))
print('Running query: {}'.format(query_string))
```

```
Glue Catalog table name: reviews-feature-group-1661803330-1661804658
Running query:
    SELECT date,
        review_id,
        sentiment,
        label_id,
        input_ids,
        review_body
    FROM "reviews-feature-group-1661803330-1661804658"
    WHERE split_type='train'
    LIMIT 5
```

Configure the S3 location for the query results. This allows us to re-use the query results for future queries if the data has not changed. We can even share this S3 location between team members to improve query performance for common queries on data that does not change often.

In [30]:
```
output_s3_uri = 's3://{}/query_results/{}/'.format(bucket, feature_store_
print(output_s3_uri)
```

```
s3://sagemaker-us-east-1-504458356961/query_results/reviews-feature-store
-1661803330/
```

Exercise 4

Query the feature store.

Instructions: Use `feature_store_query.run` function passing the constructed above query string and the location of the output S3 bucket.

```
feature_store_query.run(
    query_string=..., # query string
    output_location=... # location of the output S3 bucket
)
```

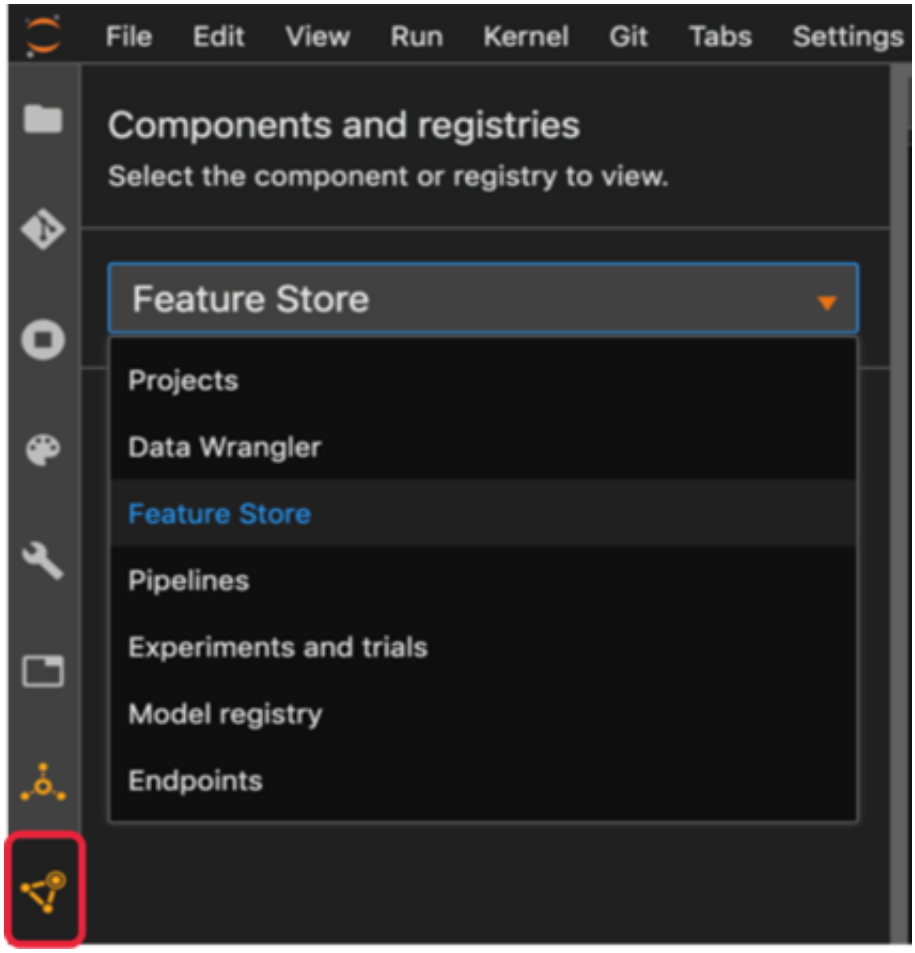
In [31]:
```
feature_store_query.run(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    query_string=query_string, # Replace None
    output_location=output_s3_uri # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
)

feature_store_query.wait()
```

In [32]:
```
import pandas as pd
pd.set_option("max_colwidth", 100)

df_feature_store = feature_store_query.as_dataframe()
df_feature_store
```

Out[32]:

	date	review_id	sentiment	label_id	input_ids	review_body
0	2022-08-29T20:24:48Z	7558	1	2	[0, 41541, 1318, 1437, 1254, 1437, 8, 939, 101, 5, 4600, 19780, 847, 36, 11655, 157, 15, 42, 308...	Nice quality details and i like the looser cut (works well on this blouse).
1	2022-08-29T20:24:48Z	3803	0	1	[0, 35703, 10, 1256, 475, 4791, 548, 6399, 14, 888, 2564, 127, 36, 3340, 45314, 43, 3065, 1368, ...	Finally a pretty maeve shirt that actually fit my (apparently) giant hulk arms.. but it's so flo...
2	2022-08-29T20:24:48Z	3316	1	2	[0, 597, 2629, 6683, 328, 2422, 3793, 8, 157, 12, 7078, 328, 657, 5, 155, 73, 306, 21764, 8, 124...	Fits perfectly! super soft and well-made! love the 3/4 sleeves and back drop. i can expect i wil...
3	2022-08-29T20:24:48Z	4699	1	2	[0, 100, 657, 42, 23204, 4, 24, 18, 3279, 1437, 3473, 1437, 8, 3793, 4, 2579, 1318, 350, 4, 2, 1...	I love this sweater. it's warm comfortable and soft. nice quality too.
4	2022-08-29T20:24:48Z	16894	0	1	[0, 100, 101, 42, 299, 53, 9574, 939, 40, 28, 3357, 4, 24, 18, 169, 350, 251, 4, 5, 3195, 16, 12...	I like this top but unfortunately i will be returning. it's way too long. the color is pretty as...

Review the Feature Store in SageMaker Studio



3.2. Export TSV from Feature Store

Save the output as a TSV file:

In [33]:
```
df_feature_store.to_csv('./feature_store_export.tsv',
                        sep='\t',
                        index=False,
                        header=True)
```

In [34]:
```
!head -n 5 ./feature_store_export.tsv
```

```
date    review_id       sentiment       label_id        input_ids
review_body
2022-08-29T20:24:48Z    7558    1       2       [0, 41541, 1318, 1437, 12
54, 1437, 8, 939, 101, 5, 4600, 19780, 847, 36, 11655, 157, 15, 42, 3089,
11556, 322, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]  Nice quality  details  and i like
the looser cut (works well on this blouse).
2022-08-29T20:24:48Z    3803    0       1       [0, 35703, 10, 1256, 475,
4791, 548, 6399, 14, 888, 2564, 127, 36, 3340, 45314, 43, 3065, 1368, 162
60, 3701, 7586, 53, 24, 18, 98, 3041, 219, 73, 32123, 219, 939, 56, 7, 67
1, 24, 4, 42, 429, 95, 28, 10, 936, 19, 145, 10, 2514, 6429, 1437, 53, 5,
7400, 10199, 156, 162, 619, 101, 939, 21, 2498, 10, 10178, 8, 156, 162, 3
56, 158, 1178, 2671, 87, 939, 21, 198, 127, 13977, 73, 7903, 1437, 190, 7
7, 9320, 66, 19, 2084, 6149, 1033, 4, 939, 802, 59, 667, 7, 1836, 159, 53
, 938, 75, 686, 114, 14, 74, 190, 6136, 5, 7400, 10199, 936, 8, 202, 989,
929, 11, 5, 3701, 4, 9, 768, 939, 64, 75, 860, 2, 1, 1, 1, 1, 1, 1, 1]
Finally a pretty maeve shirt that actually fit my (apparently) giant hulk
arms.. but it's so flowy/swingy i had to return it. this might just be a
problem with being a larger lady  but the excess fabric made me feel like
i was wearing a tent and made me look 10x bigger than i was around my wai
st/hips  even when balanced out with leggings. i thought about trying to
size down but wasn't sure if that would even solve the excess fabric prob
lem and still leave room in the arms. of course i can't try
2022-08-29T20:24:48Z    3316    1       2       [0, 597, 2629, 6683, 328,
2422, 3793, 8, 157, 12, 7078, 328, 657, 5, 155, 73, 306, 21764, 8, 124, 1
874, 4, 939, 64, 1057, 939, 40, 3568, 10, 319, 11, 42, 567, 1136, 73, 314
21, 191, 11, 6056, 4, 2200, 5940, 328, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
Fits perfectly! super soft and well-made! love the 3/4 sleeves and back d
rop. i can expect i will wear a lot in this coming fall/winter season in
ca. highly recommend!
2022-08-29T20:24:48Z    4699    1       2       [0, 100, 657, 42, 23204,
4, 24, 18, 3279, 1437, 3473, 1437, 8, 3793, 4, 2579, 1318, 350, 4, 2, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1]      I love this sweater. it's warm  comfortab
le  and soft. nice quality too.
```

Upload TSV to the S3 bucket:

In [35]:
```
!aws s3 cp ./feature_store_export.tsv s3://$bucket/feature_store/feature_
```

```
upload: ./feature_store_export.tsv to s3://sagemaker-us-east-1-5044583569
61/feature_store/feature_store_export.tsv
```

Check the file in the S3 bucket:

In [36]:
```
!aws s3 ls --recursive s3://$bucket/feature_store/feature_store_export.ts
```

```
2022-08-29 20:33:26       3665 feature_store/feature_store_export.tsv
```

3.3. Check that the dataset in the Feature Store is balanced by sentiment

Now you can setup an Athena query to check that the stored dataset is balanced by the target class `sentiment`.

Exercise 5

Write an SQL query to count the total number of the reviews per `sentiment` stored in the Feature Group.

Instructions: Pass the SQL statement of the form

```
SELECT category_column, COUNT(*) AS new_column_name
FROM table_name
GROUP BY category_column
```

into the variable `query_string_count_by_sentiment`. Here you would need to use the column `sentiment` and give a name `count_reviews` to the new column with the counts.

In [37]:
```
feature_store_query_2 = feature_group.athena_query()

# Replace all None
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
query_string_count_by_sentiment = """
SELECT sentiment, COUNT(*) AS count_reviews
FROM "{}"
GROUP BY sentiment
""".format(feature_store_table)
### END SOLUTION - DO NOT delete this comment for grading purposes
```

Exercise 6

Query the feature store.

Instructions: Use `run` function of the Feature Store query, passing the new query string `query_string_count_by_sentiment`. The output S3 bucket will remain unchanged. You can follow the example above.

In [38]:
```
feature_store_query_2.run(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    query_string=query_string_count_by_sentiment, # Replace None
    output_location=output_s3_uri # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
)

feature_store_query_2.wait()

df_count_by_sentiment = feature_store_query_2.as_dataframe()
df_count_by_sentiment
```

Out[38]:

	sentiment	count_reviews
0	0	2051
1	1	2051
2	-1	2051

Exercise 7

Visualize the result of the query in the bar plot, showing the count of the reviews by sentiment value.

Instructions: Pass the resulting data frame `df_count_by_sentiment` into the `barplot` function of the `seaborn` library.

```
sns.barplot(
    data=...,
    x='...',
    y='...',
    color="blue"
)
```

In [39]:
```
import seaborn as sns

sns.barplot(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    data=df_count_by_sentiment, # Replace None
    x='sentiment', # Replace None
    y='count_reviews', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    color="blue"
)
```

Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd1afc93dd0>
```

Upload the notebook and `prepare_data.py` file into S3 bucket for grading purposes.

Note: you may need to save the file before the upload.

In [40]:
```
!aws s3 cp ./C2_W1_Assignment.ipynb s3://$bucket/C2_W1_Assignment_Learner
!aws s3 cp ./src/prepare_data.py s3://$bucket/src/C2_W1_prepare_data_Lear
```

```
upload: ./C2_W1_Assignment.ipynb to s3://sagemaker-us-east-1-504458356961
/C2_W1_Assignment_Learner.ipynb
upload: src/prepare_data.py to s3://sagemaker-us-east-1-504458356961/src/
C2_W1_prepare_data_Learner.py
```

Please go to the main lab window and click on `Submit` button (see the `Finish the lab` section of the instructions).

In [ ]:

DeepLearning.AI

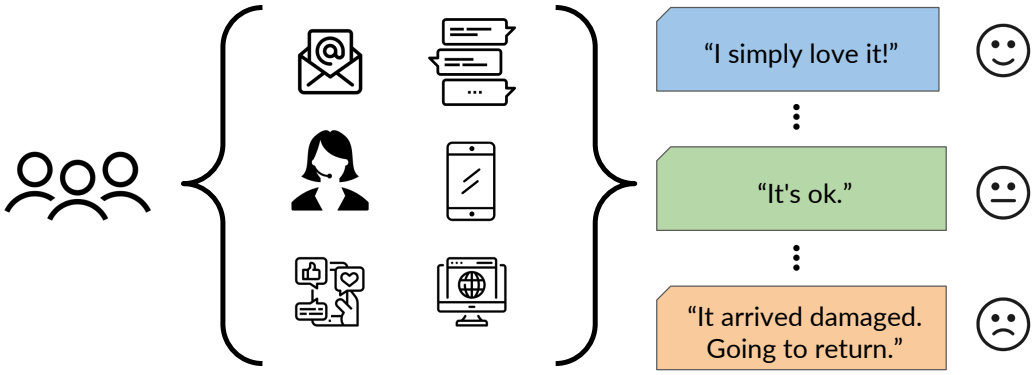
aws

Train and Debug a Custom Machine Learning Model

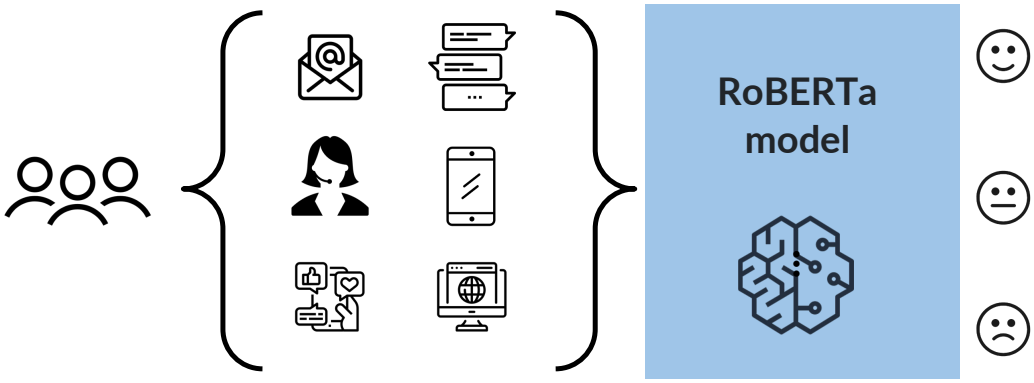
Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

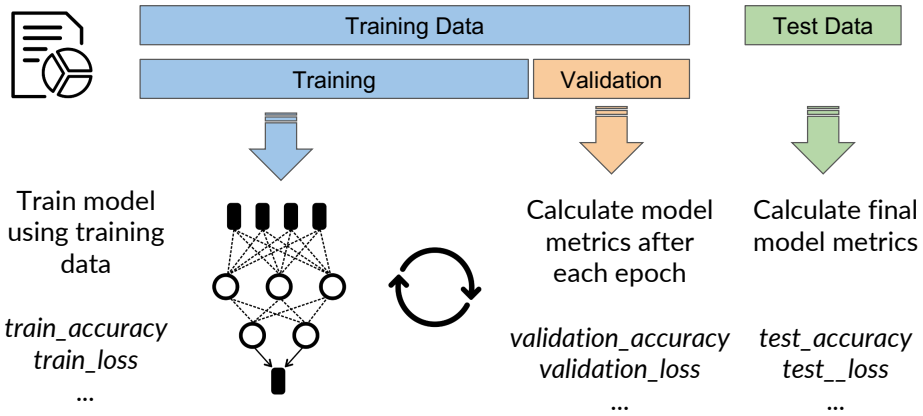
Multi-class classification for sentiment analysis of product reviews



Multi-class classification for sentiment analysis of product reviews

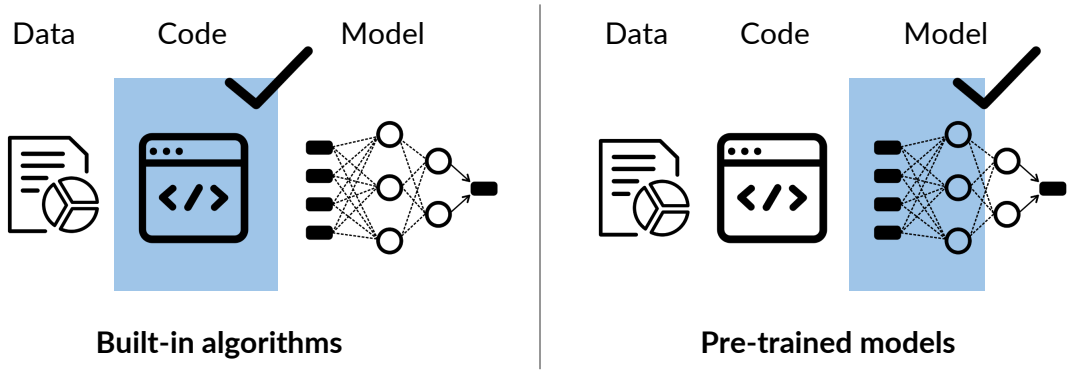


Model training - Fit the model to your data

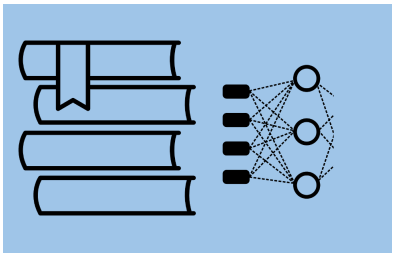


Pre-trained models

Built-in algorithms versus pre-trained models

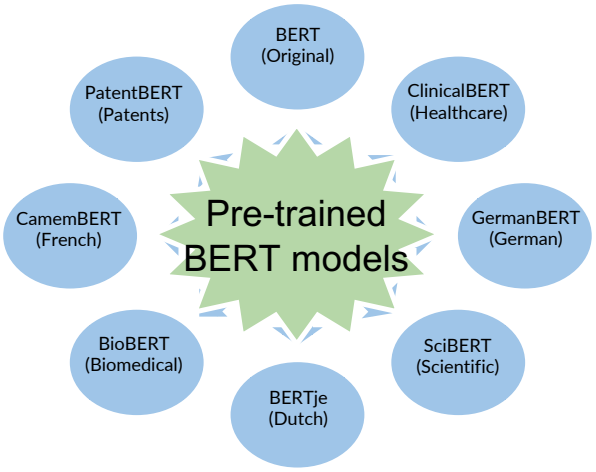


Model pre-training and fine-tuning

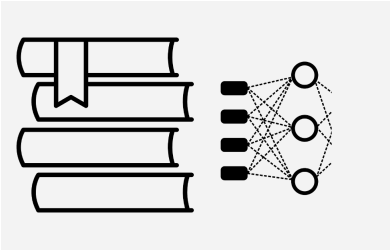


Unsupervised Learning

Pre-Training

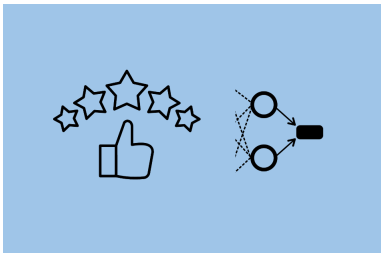


Model pre-training and fine-tuning



Unsupervised Learning

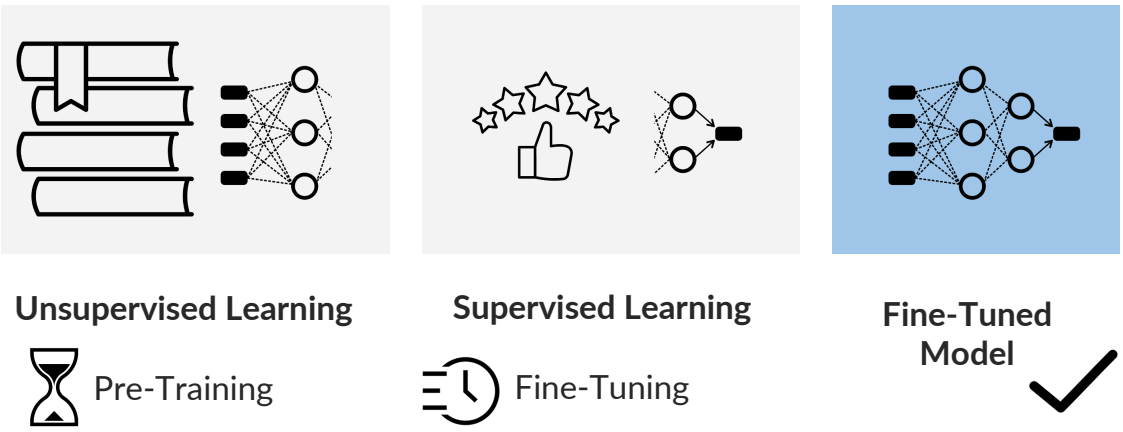
Pre-Training



Supervised Learning

Fine-Tuning

Model pre-training and fine-tuning



Where to find pre-trained models

Where to find pre-trained models

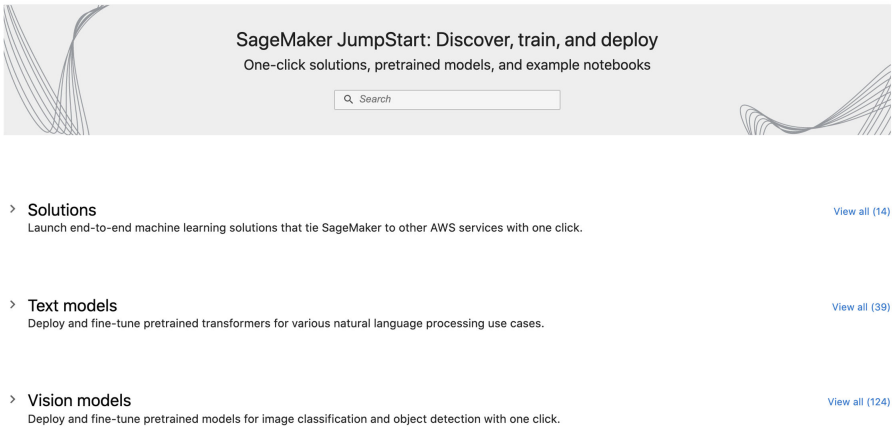
PyTorch HUB

mxnet Gluon Model Zoo

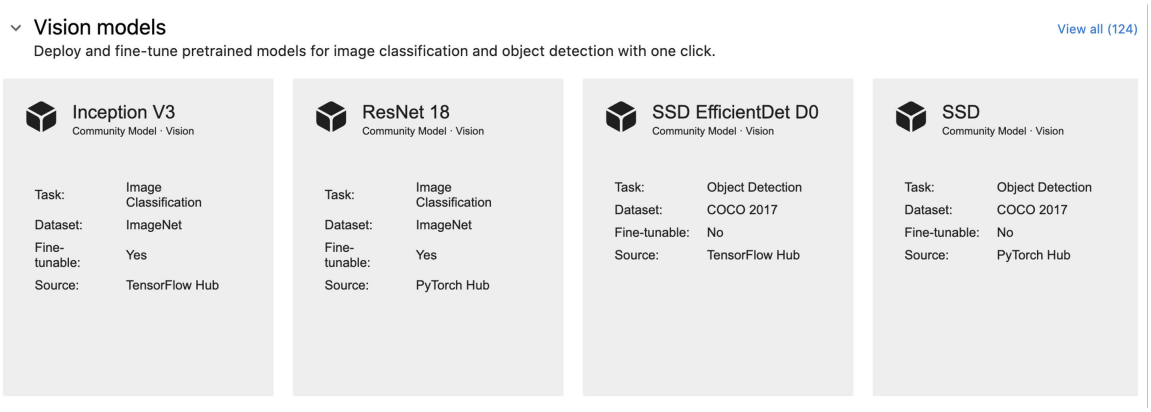
TensorFlow Hub

Hugging Face Model Hub

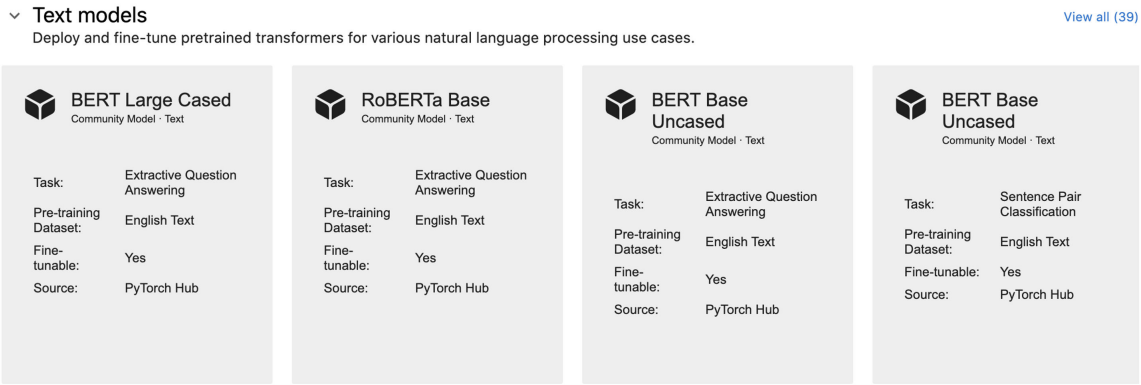
Amazon SageMaker JumpStart



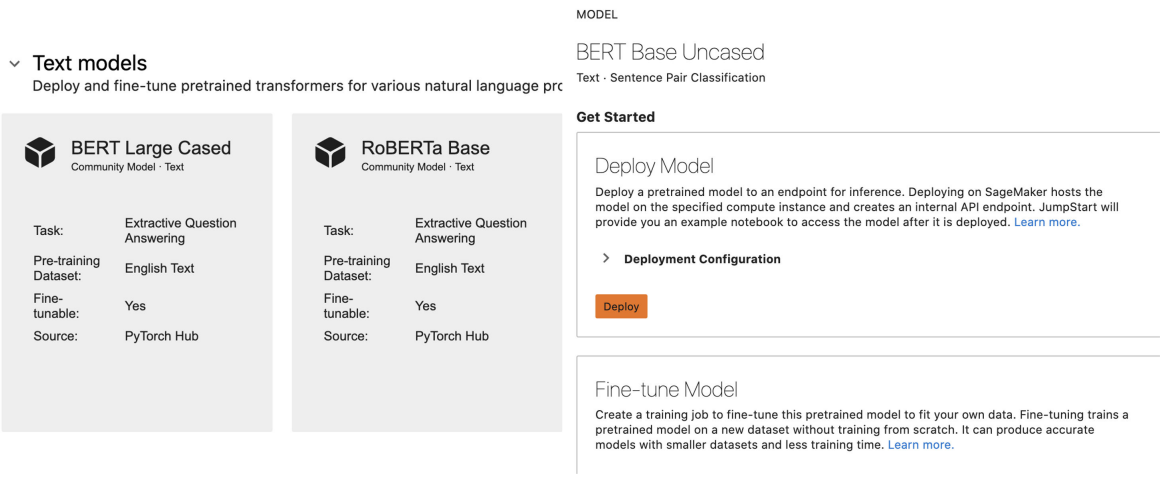
Amazon SageMaker JumpStart vision models



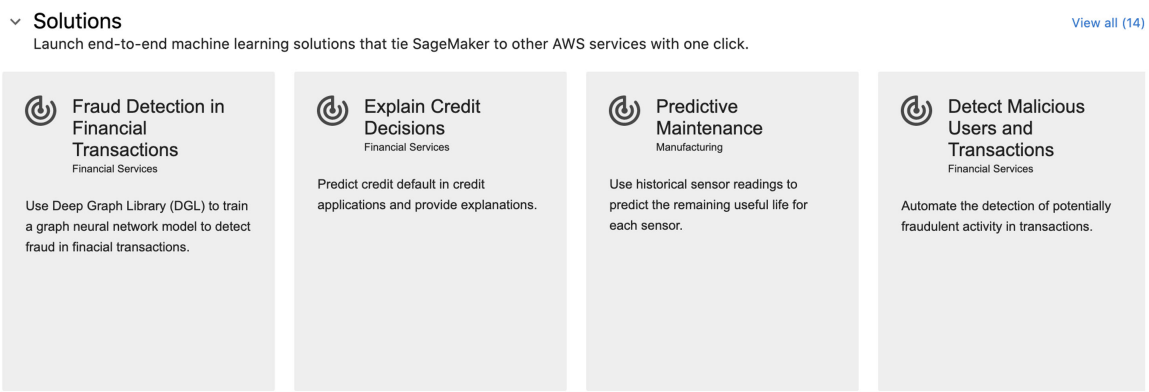
Amazon SageMaker JumpStart text models



Amazon SageMaker JumpStart text models



Amazon SageMaker JumpStart solutions



Where to find pre-trained models

PyTorch HUB

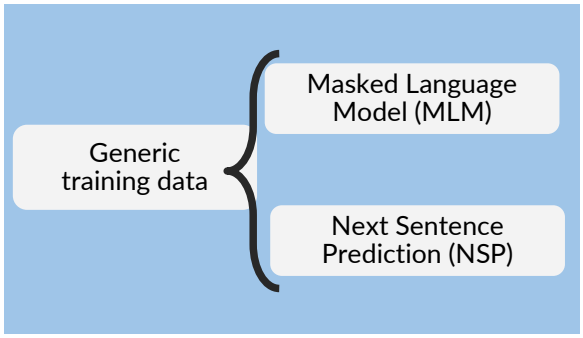
mxnet Gluon Model Zoo

TensorFlow Hub

Hugging Face Model Hub

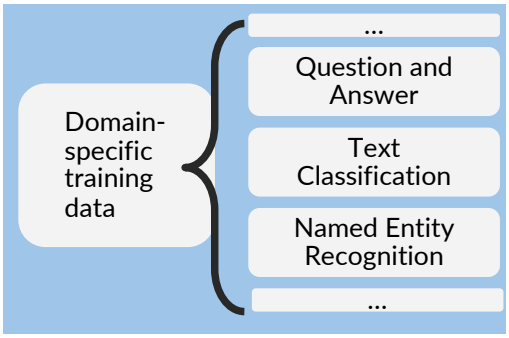
Pre-trained BERT models

BERT model pre-training and fine-tuning steps



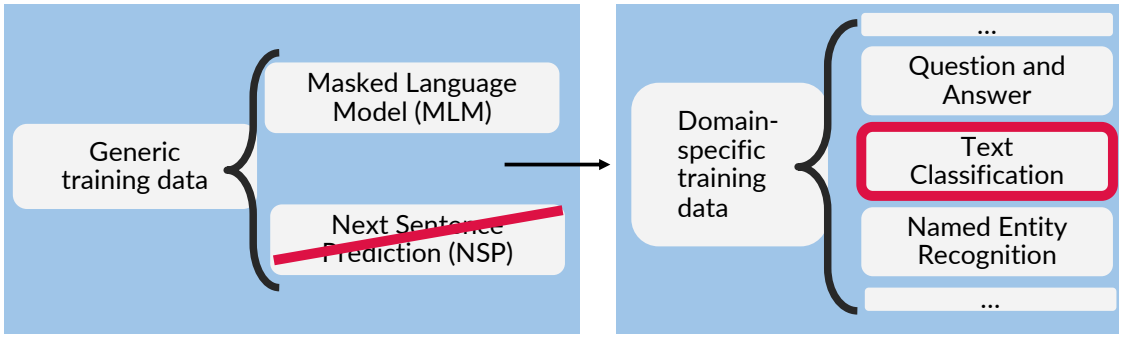
Unsupervised Learning
Pre-Training

BERT model pre-training and fine-tuning steps



Supervised Learning
Fine-Tuning

RoBERTa model



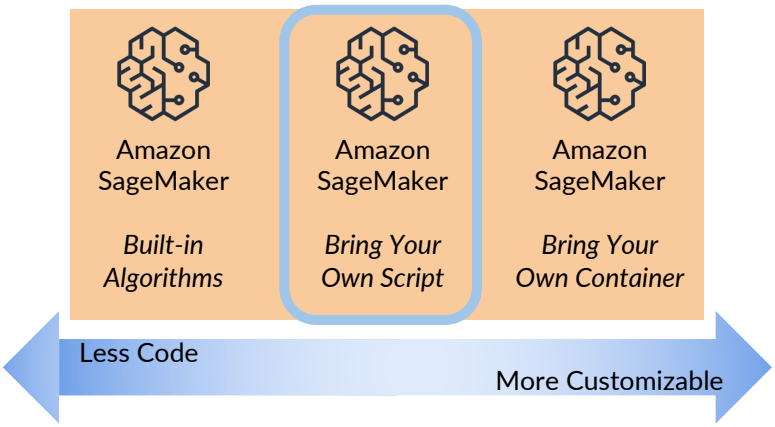
Unsupervised Learning
Pre-Training

Supervised Learning
Fine-Tuning

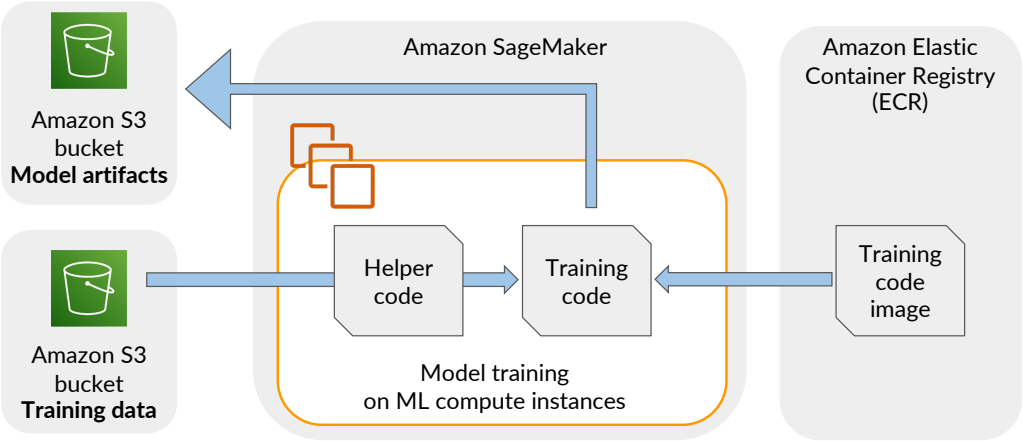
Train a custom model

with Amazon SageMaker

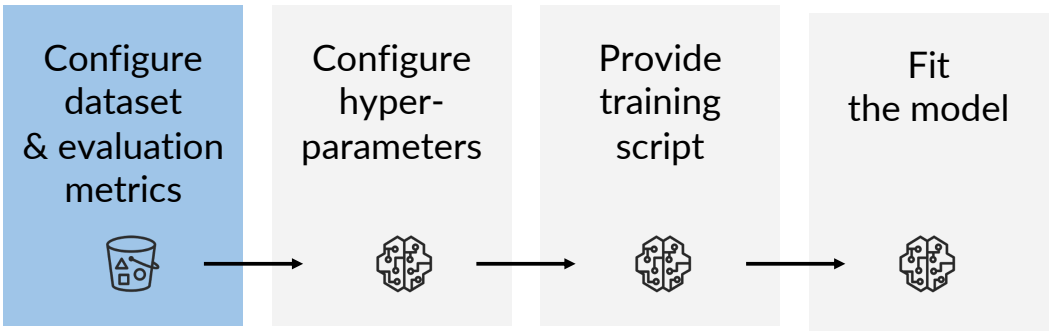
Model training options



Amazon SageMaker "Bring Your Own Script"



Steps



Configure dataset and metrics

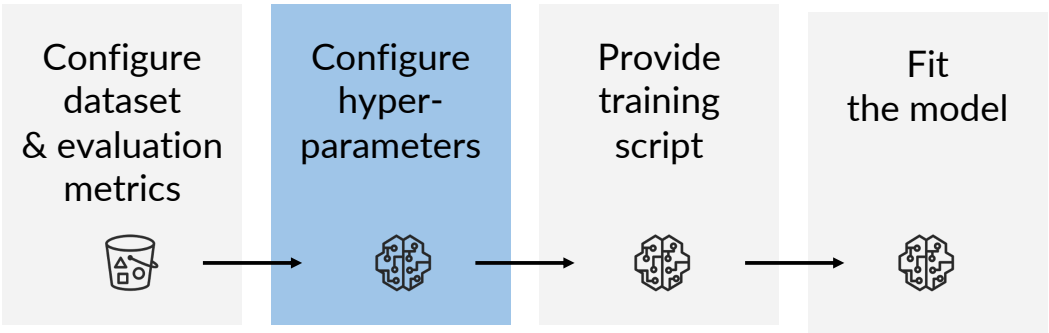
```
from sagemaker.inputs import TrainingInput

s3_input_train_data = TrainingInput(s3_data="s3://...")
s3_input_validation_data = TrainingInput(s3_data="s3://...")
s3_input_test_data = TrainingInput(s3_data="s3://...")
```

Amazon S3

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

Amazon CloudWatch Logs

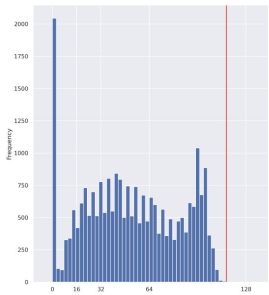
Steps



Configure model hyperparameters

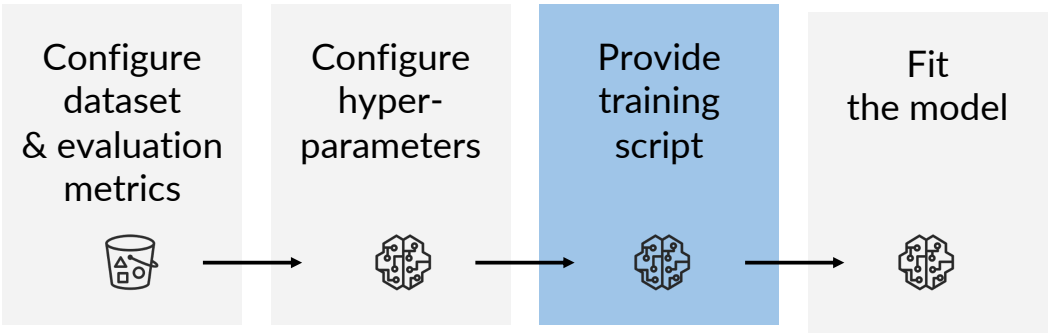
```
hyperparameters={
    'epochs': 3,
    'learning_rate': 2e-5,
    'train_batch_size': 256,
    'train_steps_per_epoch': 50,
    'validation_batch_size': 256,
    'validation_steps_per_epoch': 50,
        ...
    'max_seq_length': 128
}
```

Max. number of input tokens passed to BERT model



mean	52.51
std	31.38
min	1.00
10%	10.00
20%	22.00
30%	32.00
40%	41.00
50%	51.00
60%	61.00
70%	73.00
80%	88.00
90%	97.00
100%	**115.00**

Steps



Provide training script *src/train.py*

```
from transformers import RobertaModel, RobertaConfig
from transformers import RobertaForSequenceClassification
...

config = RobertaConfig.from_pretrained('roberta-base', num_labels=3,
                                       id2label={ 0: -1, 1: 0, 2: 1},
                                       label2id={-1: 0, 0: 1, 1: 2})
```

Import Hugging Face transformer libraries (pip install transformers)

Download model config

Specify number of labels and id2label mappings

Provide training script *src/train.py*

```
model = RobertaForSequenceClassification.from_pretrained(
                                        'roberta-base',
                                        config=config)

...

model = train_model(model, ...)
```

Download pre-trained model

Fine-tune model

Hugging Face
Model Hub

Provide training script *src/train.py*

```
def train_model(model, train_data_loader, df_train, val_data_loader, df_val, args):

    loss_function = nn.CrossEntropyLoss()
    optimizer = optim.Adam(params=model.parameters(), lr=args.learning_rate)
    ...
```

Define loss function

Creating optimizer

Provide training script *src/train.py*

```
for epoch in range(args.epochs):
    print('EPOCH -- {}'.format(epoch))
    for i, (sent, label) in enumerate(train_data_loader):
        if i < args.train_steps_per_epoch:
            model.train()
            optimizer.zero_grad()
            sent = sent.squeeze(0)
            output = model(sent)[0]
            _, predicted = torch.max(output, 1)
            loss = loss_function(output, label)
            loss.backward()
            optimizer.step()
    ...
return model
```

Loop through epochs

Put model in train mode

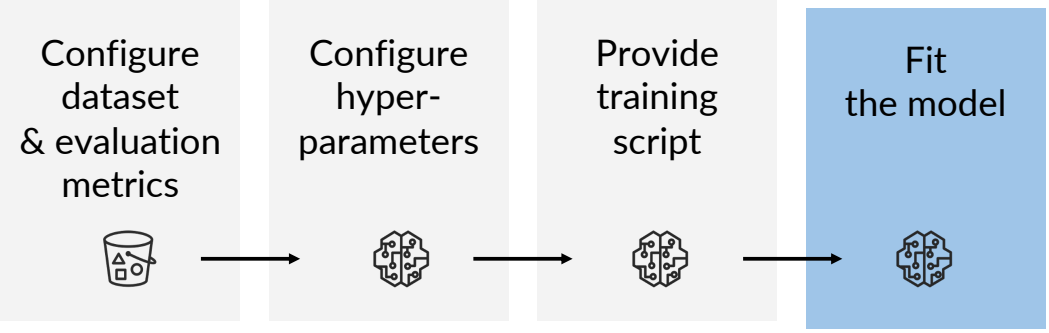
Clear gradients

Get prediction result

Compute gradients via backpropagation

Update the parameters

Steps



Fit the model

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator

estimator = PyTorchEstimator(
    entry_point='train.py',
    source_dir='src',
    role=role,
    instance_count=1,
    instance_type='ml.c5.9xlarge',
    ...
```

Model training script

Directory containing additional assets such as requirements.txt

Compute instance types and count

Fit the model

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator

estimator = PyTorchEstimator(
    ...
    framework_version=<PYTORCH_VERSION>,
    hyperparameters=hyperparameters,
    metric_definitions=metric_definitions)

estimator.fit(...)
```

PyTorch version

Pass in hyperparameters and metrics

Start model training

Debug and profile models

Debug and profile models

Detect common training errors

"Are my gradient values becoming too large or too small?"

Detect common training errors

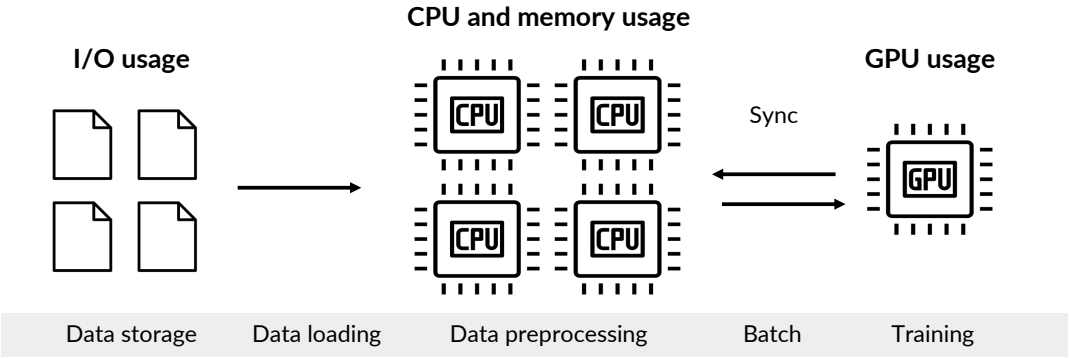
- Vanishing gradients
- Exploding gradients
- Bad initialization
- Overfitting
- ...

Debug and profile models

Detect common training errors	"Are my gradient values becoming too large or too small?"
Monitor and profile system resource utilization	"How much GPU, CPU, network, and memory does my model training consume?"

Monitor and profile system resource utilization

Potential bottlenecks:



Debug and profile models

Detect common training errors	"Are my gradient values becoming too large or too small?"
Monitor and profile system resource utilization	"How much GPU, CPU, network,and memory does my model training consume?"
Analyze errors and take action	"Stop the model training if the model starts overfitting!"

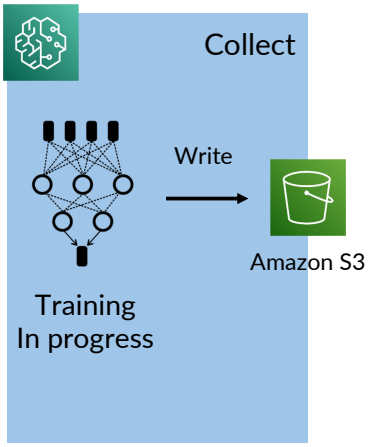
Analyze errors and take action

- Stop model training when an issue is found
- Send a notification via email an issue is found
- Send a notification via text message when an issue is found

Debug and profile models

with Amazon SageMaker Debugger

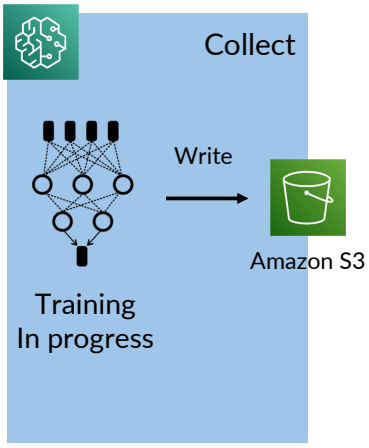
Amazon SageMaker Debugger



Capture real-time debugging data during model training in Amazon SageMaker

- System metrics
- Framework metrics
- Output tensors

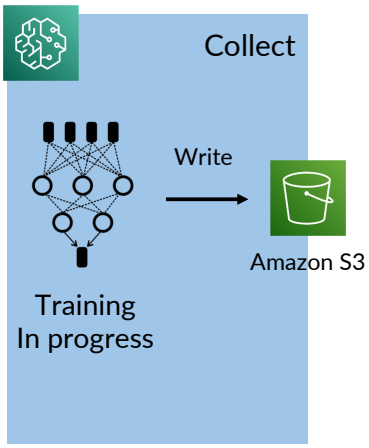
Amazon SageMaker Debugger



Capture real-time debugging data during model training in Amazon SageMaker

- System metrics
  - CPU and GPU (and memory) utilization
  - Network metrics
  - Data input and output (I/O) metrics

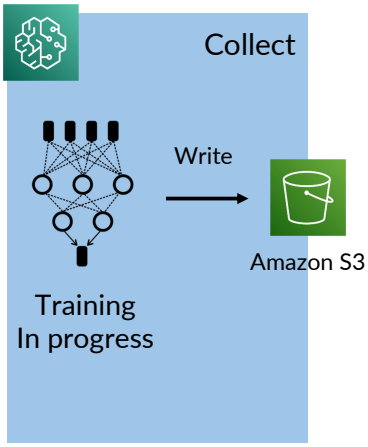
Amazon SageMaker Debugger



Capture real-time debugging data during model training in Amazon SageMaker

- Framework metrics
  - Convolutional operations in forward pass
  - Batch normalization operations in backward pass
  - Data loader processes between steps
  - Gradient descent algorithm operations

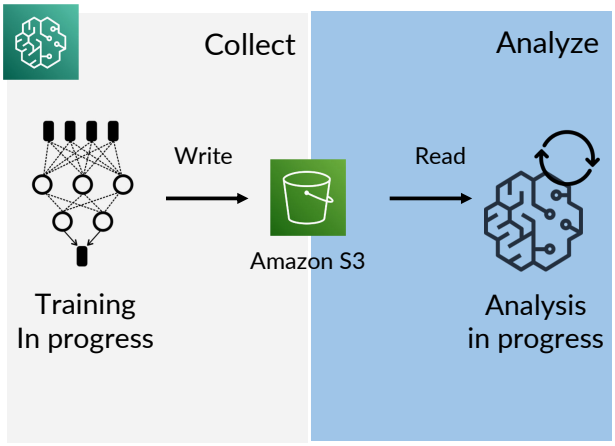
Amazon SageMaker Debugger



Capture real-time debugging data during model training in Amazon SageMaker

- Output tensors
  - Scalar values (accuracy and loss)
  - Matrices (weights, gradients, input layers, and output layers)

Amazon SageMaker Debugger



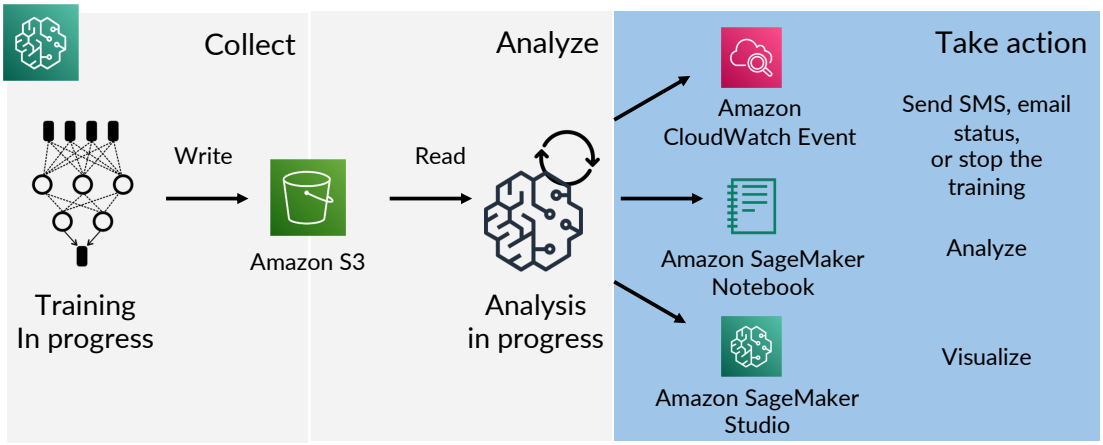
Continuous analysis through rules

Amazon SageMaker Debugger: Built-in rules

Problem class	Rules
Datasets	Class imbalance Data not normalized Ratio of tokens in sequence
Loss and accuracy	Loss not decreasing Overfitting Underfitting Overtraining Classifier confusion
Weights	Poor initialization Updates too small

Problem class	Rules
Tensor	All values zero Variance of values too small Values not changing across steps
Activation function	Tanh saturation Sigmoid saturation Dying ReLU
Decision trees	Depth of tree too large Low feature importance
Gradients	Vanishing Exploding

Amazon SageMaker Debugger



Code example

Debug model training: Configuration

```
from sagemaker.debugger import Rule, rule_configs
rules=[
    Rule.sagemaker(rule_configs.loss_not_decreasing()),
    Rule.sagemaker(rule_configs.overtraining())
    ...
]
```

Select (built-in) rules to evaluate

Overtraining prevents model from overfitting

Debug model training: Add config to estimator

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator
estimator = PyTorchEstimator(
    entry_point='train.py',
    ...
    rules=rules
)
```

Pass rules in estimator

Profile training job: Configuration

```
from sagemaker.debugger import ProfilerRule, rule_configs
rules=[
    ProfilerRule.sagemaker(rule_configs.LowGPUUtilization()),
    ProfilerRule.sagemaker(rule_configs.ProfilerReport()),
    ...]
```

Select rules for profiling

```
from sagemaker.debugger import ProfilerConfig, FrameworkProfile
profiler_config = ProfilerConfig(
    system_monitor_interval_millis=500,
    framework_profile_params=FrameworkProfile(num_steps=10))
```

Configure profiler

Profile training job: Add config to estimator

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator
estimator = PyTorchEstimator(
    entry_point='train.py', ... ,
    rules=rules,
    profiler_config=profiler_config)
```

Pass rules and config in estimator

Analyze results

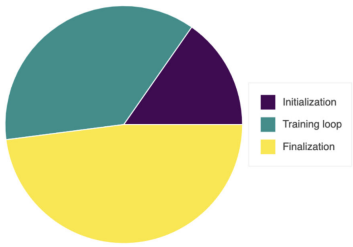
DeepLearning.AI aws

Profiling Report

Training job summary

The following table gives a summary about the training job. The table includes information about when the training job started and ended, how much time initialization, training loop and finalization took. Your training job started on 05/06/2021 at 17:53:36 and ran for 323 seconds.

#		Job Statistics
0	Start time	17:53:36 05/06/2021
1	End time	17:58:59 05/06/2021
2	Job duration	323 seconds
3	Training loop start	17:54:26 05/06/2021
4	Training loop end	17:56:26 05/06/2021
5	Training loop duration	119 seconds
6	Initialization time	49 seconds
7	Finalization time	153 seconds
8	Initialization	15 %
9	Training loop	36 %
10	Finalization	47 %



DeepLearning.AI aws

Profiling Report - Rules summary

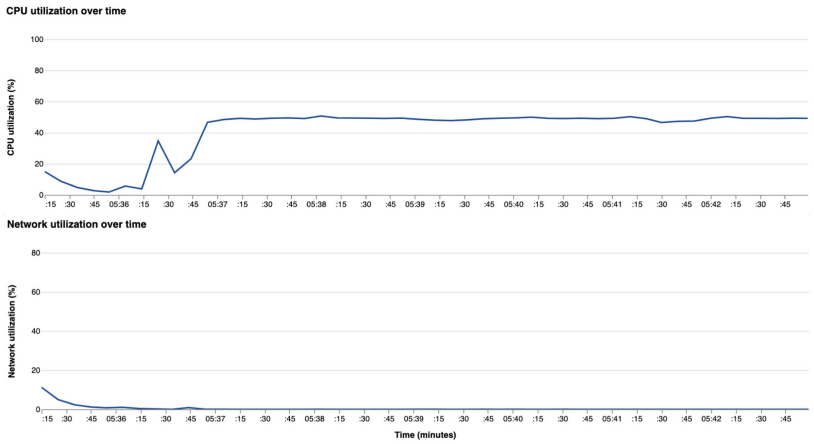
Rules summary

The following table shows a profiling summary of the Debugger built-in rules. The table is sorted by the rules that triggered the most frequently. During your training job, the MaxInitializationTime rule was the most frequently triggered. It processed 295 datapoints and was triggered 0 times.

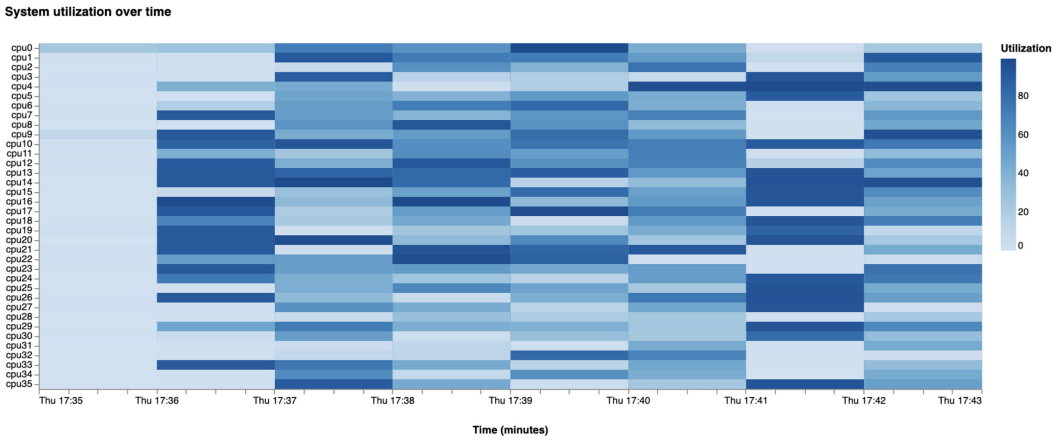
	Description	Recommendation	Number of times rule triggered	Number of datapoints	Rule parameters
MaxInitializationTime	Checks if the time spent on initialization exceeds a threshold percent of the total training time. The rule waits until the first step of training loop starts. The initialization can take longer if downloading the entire dataset from Amazon S3 in File mode. The default threshold is 20 minutes.	Initialization takes too long. If using File mode, consider switching to Pipe mode in case you are using TensorFlow framework.	0	295	threshold:20
GPUMemoryIncrease	Measures the average GPU memory footprint and triggers if there is a large increase.	Choose a larger instance type with more memory if footprint is close to maximum available memory.	0	0	increase:5 patience:1000 window:10
StepOutlier	Detects outliers in step duration. The step duration for forward and backward pass should be roughly the same throughout the training. If there are significant outliers, it may indicate a system stall or bottleneck issues.	Check if there are any bottlenecks (CPU, I/O) correlated to the step outliers.	0	295	threshold:3 mode:None n_outliers:10 stddev:3
BatchSize	Checks if GPUs are underutilized because the batch size is too small. To detect this problem, the rule analyzes the average GPU memory footprint, the CPU and the GPU utilization.	The batch size is too small, and GPUs are underutilized. Consider running on a smaller instance type or increasing the batch size.	0	930	cpu_threshold_p95:70 gpu_threshold_p95:70 gpu_memory_threshold_p95:70 patience:1000 window:500

DeepLearning.AI aws

Profiling Report - CPU/network utilization



Profiling Report - System utilization



Train a review classifier with BERT and Amazon SageMaker

Introduction

In the previous lab you performed Feature Engineering on the raw dataset, preparing it for training the model. Now you will train a text classifier using a variant of BERT called RoBERTa - a Robustly Optimized BERT Pretraining Approach - within a PyTorch model ran as a SageMaker Training Job.

Table of Contents

- 1. Configure dataset, hyper-parameters and evaluation metrics
  - 1.1. Configure dataset
    - Exercise 1
    - Exercise 2
    - Exercise 3
  - 1.2. Configure model hyper-parameters
  - 1.3. Setup evaluation metrics
  - 1.4. Setup Debugger and Profiler
- 2. Train model
  - 2.1. Setup the RoBERTa and PyTorch script to run on SageMaker
    - Exercise 4
    - Exercise 5
    - Exercise 6
  - 2.2. Analyze Debugger results
  - 2.3. Download SageMaker debugger profiling report
- 3. Deploy the model
- 4. Test model

Let's review Amazon SageMaker "Bring Your Own Script" scheme:

In this lab you will cover each part of the scheme. First, 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
!conda install -q -y pytorch==1.6.0 -c pytorch
!pip install --disable-pip-version-check -q transformers==3.5.1
```

```
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... failed with initial frozen solve. Retr
ying with flexible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: /opt/conda

  added / updated specs:
    - pytorch==1.6.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2022.07.19 |       h06a4308_0         124 KB
    conda-4.14.0               |   py37h06a4308_0         909 KB
    cudatoolkit-10.2.89        |       hfd86e86_1       365.1 MB
    ninja-1.10.2               |       h06a4308_5           8 KB
    ninja-base-1.10.2          |       hd09550d_5         109 KB
    pytorch-1.6.0              |py3.7_cuda10.2.89_cudnn7.6.5_0       537.
7 MB  pytorch
    ------------------------------------------------------------
                                           Total:       903.9 MB

The following NEW packages will be INSTALLED:

  cudatoolkit        pkgs/main/linux-64::cudatoolkit-10.2.89-hfd86e86_1
  ninja              pkgs/main/linux-64::ninja-1.10.2-h06a4308_5
  ninja-base         pkgs/main/linux-64::ninja-base-1.10.2-hd09550d_5
  pytorch            pytorch/linux-64::pytorch-1.6.0-py3.7_cuda10.2.89_cu
dnn7.6.5_0

The following packages will be UPDATED:

  ca-certificates    conda-forge::ca-certificates-2022.6.1~ --> pkgs/main
::ca-certificates-2022.07.19-h06a4308_0

The following packages will be SUPERSEDED by a higher-priority channel:

  conda              conda-forge::conda-4.14.0-py37h89c186~ --> pkgs/main
::conda-4.14.0-py37h06a4308_0


Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
Retrieving notices: ...working... done
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
import numpy as np
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c2/w2')

# 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
```

In [4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

1. Configure dataset, hyper-parameters and evaluation metrics

1.1. Configure dataset

You have already transformed and balanced the data into a format that the model expects. Let's copy this data to S3. You will be using training and validation datasets to train the model. Test dataset will be used for tuning later. Setup the paths:

In [5]:
```
processed_train_data_s3_uri = 's3://{}/data/sentiment-train/'.format(buck
processed_validation_data_s3_uri = 's3://{}/data/sentiment-validation/'.f
```

Upload the data to S3 bucket:

In [6]:
```
!aws s3 cp --recursive ./data/sentiment-train $processed_train_data_s3_ur
!aws s3 cp --recursive ./data/sentiment-validation $processed_validation_
```

```
upload: data/sentiment-train/part-algo-1-womens_clothing_ecommerce_review
s.tsv to s3://sagemaker-us-east-1-653310840482/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-1-653310840482/data/sentiment-valida
tion/part-algo-1-womens_clothing_ecommerce_reviews.tsv
```

Check the existence of those files in the S3 bucket:

In [7]:
```
!aws s3 ls --recursive $processed_train_data_s3_uri
```

```
2022-09-04 02:54:11    4894416 data/sentiment-train/part-algo-1-womens_cl
othing_ecommerce_reviews.tsv
```

In [8]:
```
!aws s3 ls --recursive $processed_validation_data_s3_uri
```

```
2022-09-04 02:54:12     276522 data/sentiment-validation/part-algo-1-wome
ns_clothing_ecommerce_reviews.tsv
```

You will need to setup the input data channels, wrapping the S3 locations in a `TrainingInput` object to use with the SageMaker Training Job. This can be organized as a dictionary

```
data_channels = {
    'train': ..., # training data
    'validation': ... # validation data
}
```

where training and validation data are the Amazon SageMaker channels for S3 input data sources.

Exercise 1

Create a train data channel.

Instructions: Pass the S3 input path for training data into the `sagemaker.inputs.TrainingInput` function.

In [9]:
```
s3_input_train_data = sagemaker.inputs.TrainingInput(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    s3_data=processed_train_data_s3_uri # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
)
```

Exercise 2

Create a validation data channel.

Instructions: Pass the S3 input path for validation data into the `sagemaker.inputs.TrainingInput` function.

In [10]:
```
s3_input_validation_data = sagemaker.inputs.TrainingInput(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    s3_data=processed_validation_data_s3_uri # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
)
```

Exercise 3

Organize data channels defined above as a dictionary.

In [11]:
```
data_channels = {
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    'train': s3_input_train_data, # Replace None
    'validation': s3_input_validation_data # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
}
```

1.2. Configure model hyper-parameters

Set the Training Job parameters including the instance type, instance count, learning rate, batch size 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 [12]:
```
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
learning_rate=2e-5
train_batch_size=256
train_steps_per_epoch=50
validation_batch_size=256
validation_steps_per_epoch=50
seed=42
run_validation=True

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

Some of them will be passed into the PyTorch estimator in the hyperparameters argument. Let's setup the dictionary for that:

In [13]:
```
hyperparameters={
    'max_seq_length': max_seq_length,
    'freeze_bert_layer': freeze_bert_layer,
    'epochs': epochs,
    'learning_rate': learning_rate,
    'train_batch_size': train_batch_size,
    '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
}
```

1.3. Setup 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 [14]:
```
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

1.4. Setup Debugger and Profiler

Amazon SageMaker Debugger can be used to profile machine learning models, helping to identify and fix training issues caused by hardware resource usage. Setting some parameters in the SageMaker estimator, without any change to the training code, you can enable the collection of infrastructure and model metrics such as: CPU and GPU, RAM and GPU RAM, data loading time, time spent in ML operators running on CPU and GPU, distributed training metrics and many more. In addition, you can visualize how much time is spent in different phases, such as preprocessing, training loop, and postprocessing. If needed, you can drill down on each training epoch, and even on each function in your training script.

Define Debugger Rules as described here:
https://docs.aws.amazon.com/sagemaker/latest/dg/debugger-built-in-rules.html

In [15]:
```
from sagemaker.debugger import Rule, ProfilerRule, rule_configs
from sagemaker.debugger import DebuggerHookConfig
from sagemaker.debugger import ProfilerConfig, FrameworkProfile
```

`DebuggerHookConfig` provides options to customize how debugging information is emitted and saved. `s3_output_path` argument value defines the location in Amazon S3 to store the output.

In [16]:
```
debugger_hook_config = DebuggerHookConfig(
    s3_output_path='s3://{}'.format(bucket),
)
```

`ProfilerConfig` sets the configuration for collecting system and framework metrics of SageMaker Training Jobs. Parameter `system_monitor_interval_millis` sets the time interval to collect system metrics (in milliseconds). Parameter `framework_profile_params` is the object for framework metrics profiling. Here you will set its local path, the step at which to start profiling, `start_step`, and the number of steps to profile, `num_steps`.

In [17]:
```
from sagemaker.debugger import ProfilerConfig, FrameworkProfile

profiler_config = ProfilerConfig(
    system_monitor_interval_millis=500,
    framework_profile_params=FrameworkProfile(local_path="/opt/ml/output/
)
```

For monitoring and profiling the built-in rules you can use the `ProfilerReport`. It creates a profiling report and updates when the individual rules are triggered. If you trigger this `ProfilerReport` rule without any customized parameter as in the cell below, then the `ProfilerReport` rule triggers all of the built-in rules for monitoring and profiling with their default parameter values.

The profiling report can be downloaded while the Training Job is running or after the job has finished.

In [18]:
```
rules=[ProfilerRule.sagemaker(rule_configs.ProfilerReport())]
```

2. Train model

2.1. Setup the RoBERTa and PyTorch script to run on SageMaker

You will prepare the PyTorch model to run as a SageMaker Training Job in a separate Python file, which will be called during the training.

Here you will be using the pre-trained model `roberta-base`. The information about the available models can be found in the Hugging Face website.

Exercise 4

1. Open the file src/train.py. Go through the comments to understand its content.
2. Find and review the `configure_model()` function, which contains the RoBERTa model configuration.
3. In the following function investigate given mapping `label2id` of a 0-indexed list of classes used by BERT [0, 1, 2] to the list of the sentiment values [-1, 0, 1]:

```
config = RobertaConfig.from_pretrained(
        PRE_TRAINED_MODEL_NAME,
        num_labels=len(classes),
        id2label={
            ...: ...,
            ...: ...,
            ...: ...,
        },
        label2id={
            -1: 0,
            0: 1,
            1: 2,
        }
    )
```

1. Update the function setting up the opposite mapping `id2label`: sentiment values [-1, 0, 1] to a 0-indexed list of classes used by BERT.

2. Save the file src/train.py (with the menu command File -> Save Python File).

In [19]:
```
import sys, importlib
sys.path.append('src/')

import train

# reload the module if it has been previously loaded
if 'train' in sys.modules:
    importlib.reload(train)

# Ignore warnings below
config = train.configure_model()

label_0 = config.id2label[0]
label_1 = config.id2label[1]
label_2 = config.id2label[2]

updated_correctly = False

if label_0 != -1 or label_1 != 0 or label_2 != 1:
    print('##############################################################
    print('Please check that the function \'configure_model\' in the file
    print('##############################################################
    raise Exception('Please check that the function \'configure_model\' i
else:
    print('##################')
    print('Updated correctly!')
    print('##################')

    updated_correctly = True
```

```
HBox(children=(FloatProgress(value=0.0, description='Downloading', max=48
1.0, style=ProgressStyle(description_…
##################
Updated correctly!
##################
```

Setup the PyTorch estimator to train our model. For more information on the PyTorch estimator, see the documentation here.

In [20]:
```
from sagemaker.pytorch import PyTorch as PyTorchEstimator

if updated_correctly:
    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', # dynamically retrieves the correct training im
        framework_version='1.6.0', # dynamically retrieves the correct tr
        hyperparameters=hyperparameters,
        metric_definitions=metric_definitions,
        input_mode=input_mode,
        debugger_hook_config=debugger_hook_config,
        profiler_config=profiler_config,
        rules=rules
    )
```

Exercise 5

Launch the SageMaker Training Job which will be fitting the model to the dataset.

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

```
estimator.fit(
    inputs=..., # train and validation input
    wait=False # do not wait for the job to complete before
continuing
)
```

In [21]:
```
estimator.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
    wait=False
)
```

You can refer to the last Training Job using the estimator function `latest_training_job`. Then the Training Job name can be found with the `name` function:

In [22]:
```
training_job_name = estimator.latest_training_job.name

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

```
Training Job name: pytorch-training-2022-09-04-02-57-28-212
```

You can also load the information about the Training Job using the function `describe()`. The result is in dictionary format. Let's check that it has the same Training Job name:

In [23]:
```
training_job_name = estimator.latest_training_job.describe()['TrainingJob

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

```
Training Job name: pytorch-training-2022-09-04-02-57-28-212
```

Exercise 6

Pull the Training Job status from the Training Job description.

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

In [24]:
```
print(estimator.latest_training_job.describe().keys())
```

```
dict_keys(['TrainingJobName', 'TrainingJobArn', 'TrainingJobStatus', 'Sec
ondaryStatus', 'HyperParameters', 'AlgorithmSpecification', 'RoleArn', 'I
nputDataConfig', 'OutputDataConfig', 'ResourceConfig', 'StoppingCondition
', 'CreationTime', 'LastModifiedTime', 'SecondaryStatusTransitions', 'Ena
bleNetworkIsolation', 'EnableInterContainerTrafficEncryption', 'EnableMan
agedSpotTraining', 'DebugHookConfig', 'ProfilerConfig', 'ProfilerRuleConf
igurations', 'ProfilerRuleEvaluationStatuses', 'ProfilingStatus', 'Respon
seMetadata'])
```

In [25]:

```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
training_job_status_primary = estimator.latest_training_job.describe()['T
### END SOLUTION - DO NOT delete this comment for grading purposes
print('Training Job status: {}'.format(training_job_status_primary))
```

```
Training Job status: InProgress
```

Review the Training Job in the console.

Instructions:

- open the link
- notice that you are in the section `Amazon SageMaker` -> `Training jobs`
- check the name of the Training Job, its status and other available information
- review metrics in the `Monitor` section

In [26]:

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

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

Review Training Job

Review the Cloud Watch logs (after about 5 minutes).

Instructions:

- open the link
- open the log stream with the name, which starts from the training job name
- have a quick look at the log messages

In [27]:

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

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

Review CloudWatch logs after about 5 minutes

In [28]:

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

display(HTML('<b>Review <a target="blank" href="https://s3.console.aws.am
```

Review S3 output data after the Training Job has completed

Wait for the Training Job to complete.

This cell will take approximately 30-40 minutes to run.

In [29]:
```
%%time

estimator.latest_training_job.wait(logs=False)
```

```
2022-09-04 02:58:12 Starting - Preparing the instances for training.....
2022-09-04 02:58:49 Downloading - Downloading input data...
2022-09-04 02:59:05 Training - Downloading the training image......
2022-09-04 02:59:40 Training - Training image download completed. Trainin
g in progress.....................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
................
2022-09-04 03:43:06 Uploading - Uploading generated training model.......
.............................
2022-09-04 03:46:23 Completed - Training job completed
CPU times: user 2.4 s, sys: 334 ms, total: 2.74 s
Wall time: 48min 3s
```

Wait until the ^^ Training Job ^^ completes above

Review the training metrics.

In [30]:
```
df_metrics = estimator.training_job_analytics.dataframe()
df_metrics
```

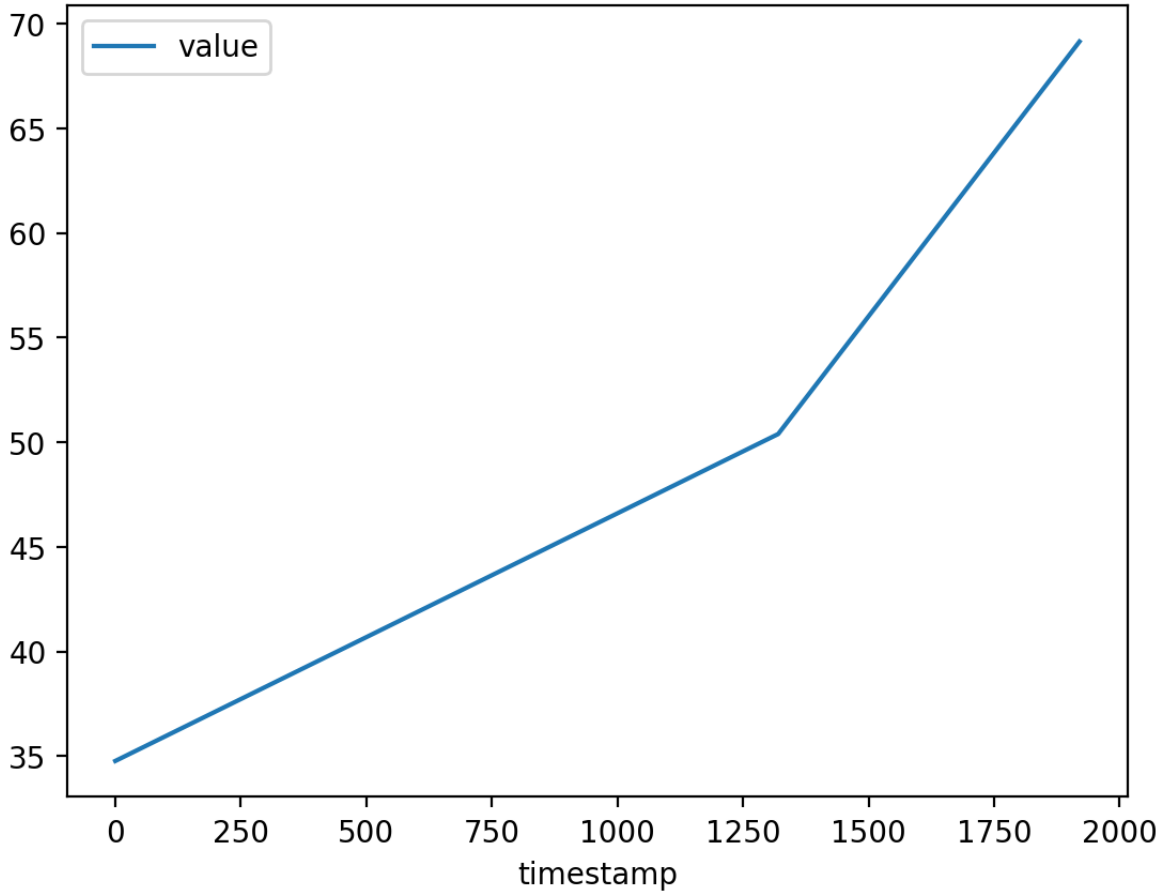
Out[30]:

	timestamp	metric_name	value
0	0.0	validation:loss	1.10
1	1320.0	validation:loss	1.02
2	1920.0	validation:loss	0.66
3	0.0	validation:accuracy	34.77
4	1320.0	validation:accuracy	50.39
5	1920.0	validation:accuracy	69.14

You can query and plot the training metrics:

In [31]:
```
df_metrics.query("metric_name=='validation:accuracy'").plot(x='timestamp'
```

Out[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5abc8c3690>
```



2.2. Analyze Debugger results

You can now explore the debugger output data:

In [32]:
```
from IPython.core.display import display, HTML

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

Review S3 debugger output data

2.3. Download SageMaker debugger profiling report

You can download and review the debugger profiling report. Here is the path in the S3 bucket:

In [33]:
```
profiler_report_s3_uri = "s3://{}/{}/rule-output/ProfilerReport/profiler-
```

You can list the report files:

In [34]:
```
!aws s3 ls $profiler_report_s3_uri/
```

```
                           PRE profiler-reports/
2022-09-04 03:46:29     365114 profiler-report.html
2022-09-04 03:46:29     212170 profiler-report.ipynb
```

The folder `profiler-reports` contains the built-in rule analysis components, stored in JSON and a Jupyter notebook. They are aggregated into the report.

In [35]:
```
!aws s3 cp --recursive $profiler_report_s3_uri ./profiler_report/
```

```
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/GPUMemoryIncrease.json to profiler_report/profiler-reports/GPUMemoryIn
crease.json
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
t.ipynb to profiler_report/profiler-report.ipynb
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/Dataloader.json to profiler_report/profiler-reports/Dataloader.json
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
t.html to profiler_report/profiler-report.html
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/LoadBalancing.json to profiler_report/profiler-reports/LoadBalancing.j
son
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/IOBottleneck.json to profiler_report/profiler-reports/IOBottleneck.jso
n
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/BatchSize.json to profiler_report/profiler-reports/BatchSize.json
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/CPUBottleneck.json to profiler_report/profiler-reports/CPUBottleneck.j
son
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/LowGPUUtilization.json to profiler_report/profiler-reports/LowGPUUtili
zation.json
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/MaxInitializationTime.json to profiler_report/profiler-reports/MaxInit
ializationTime.json
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/OverallFrameworkMetrics.json to profiler_report/profiler-reports/Overa
llFrameworkMetrics.json
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/OverallSystemUsage.json to profiler_report/profiler-reports/OverallSys
temUsage.json
download: s3://sagemaker-us-east-1-653310840482/pytorch-training-2022-09-
04-02-57-28-212/rule-output/ProfilerReport/profiler-output/profiler-repor
ts/StepOutlier.json to profiler_report/profiler-reports/StepOutlier.json
```

You can review the profiler report in the console.

Note: Click `Trust HTML` in the profiler-report.html tab that opens (on top of the document).

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

display(HTML('<b>Review <a target="blank" href="./profiler_report/profile
```

Review profiler report

3. Deploy the model

Create a custom `SentimentPredictor` that encapsulates a JSONLines serializer and deserializer. To be passed into the `PyTorchModel` it needs to be wrapped as a class.

In [37]:
```
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,
                         serializer=JSONLinesSerializer(),
                         deserializer=JSONLinesDeserializer())
```

In [38]:
```
import time
from sagemaker.pytorch.model import PyTorchModel

timestamp = int(time.time())

pytorch_model_name = '{}-{}-{}'.format(training_job_name, 'pt', timestamp

model = PyTorchModel(name=pytorch_model_name,
                     model_data=estimator.model_data,
                     predictor_cls=SentimentPredictor,
                     entry_point='inference.py',
                     source_dir='src',
                     framework_version='1.6.0',
                     py_version='py3',
                     role=role)
```

In [39]:
```
import time

pytorch_endpoint_name = '{}-{}-{}'.format(training_job_name, 'pt', timest

print(pytorch_endpoint_name)
```

```
pytorch-training-2022-09-04-02-57-28-212-pt-1662263263
```

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

In [40]:
```
%%time

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

```
----------!CPU times: user 2min 15s, sys: 9.97 s, total: 2min 25s
Wall time: 7min 27s
```

Review the Endpoint in the AWS console.

Instructions:

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

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

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

Review SageMaker REST Endpoint

4. Test model

Here, we will pass sample strings of text to the endpoint in order to see the sentiment. We give you one example of each, however, feel free to play around and change the strings yourself!

In [42]:
```
inputs = [
    {"features": ["I love this product!"]},
    {"features": ["OK, but not great."]},
    {"features": ["This is not the right product."]},
]

predictor = SentimentPredictor(endpoint_name=pytorch_endpoint_name,
                               sagemaker_session=sess)

predicted_classes = predictor.predict(inputs)

for predicted_class in predicted_classes:
    print("Predicted class {} with probability {}".format(predicted_class
```

```
Predicted class 1 with probability 0.9605445861816406
Predicted class 0 with probability 0.5798221230506897
Predicted class -1 with probability 0.7667604684829712
```

Upload the notebook and train.py file into S3 bucket for grading purposes.

Note: you may need to save the file before the upload.

In [43]:
```
!aws s3 cp ./C2_W2_Assignment.ipynb s3://$bucket/C2_W2_Assignment_Learner
!aws s3 cp ./src/train.py s3://$bucket/src/C2_W2_train_Learner.py
```

```
upload: ./C2_W2_Assignment.ipynb to s3://sagemaker-us-east-1-653310840482
/C2_W2_Assignment_Learner.ipynb
upload: src/train.py to s3://sagemaker-us-east-1-653310840482/src/C2_W2_t
rain_Learner.py
```

Please go to the main lab window and click on `Submit` button (see the `Finish the lab` section of the instructions).

In [ ]:

Machine Learning Operations (MLOps)

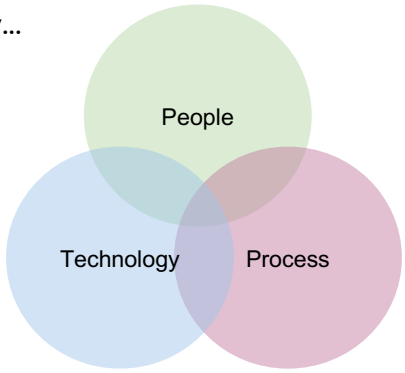
Overview

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Path to production for ML models

It's not just **technology**...



Path to production for ML models

Considerations

- Machine Learning Development Lifecycle (MLDC) != Software Development Lifecycle (SDLC)
- A Model may be a small part of an overall solution
- Multiple personas spanning the MLDC
- Integration with traditional IT practices

Path to production for ML models

Challenges

- Culture & Lack of cross functional teams
- Integration into client applications & existing tooling
- Multiple disparate pipelines & dependencies (ex. Code, Data, Training)

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines

Operationalizing Machine Learning

Goals

Accelerate the path to production:

- ❑ Reduce manual hand-offs between steps
- ❑ Increase automation within steps
- ❑ Orchestrate the workflow

Improve the quality of deployed models:

- ❑ Implement automated workflows with quality gates

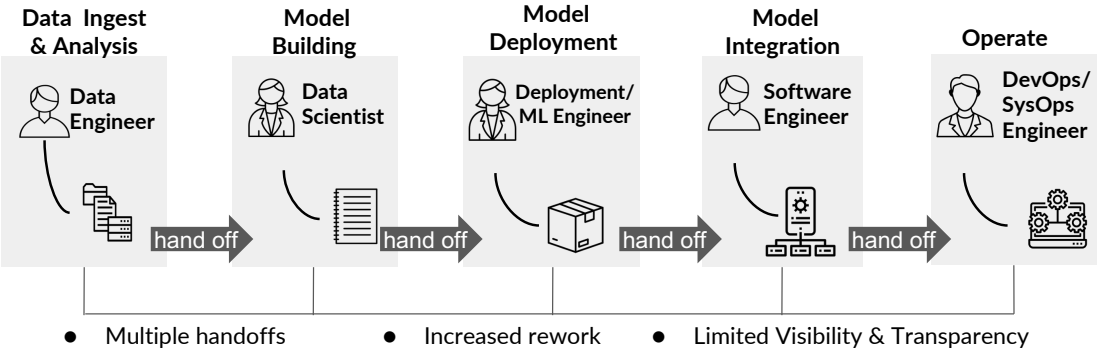
Build resilient, secure, performant, operationally efficient and cost optimized ML solutions

- ❑ Consider aspects unique to ML solutions + Traditional systems engineering considerations

Operationalizing Machine Learning

Path to production

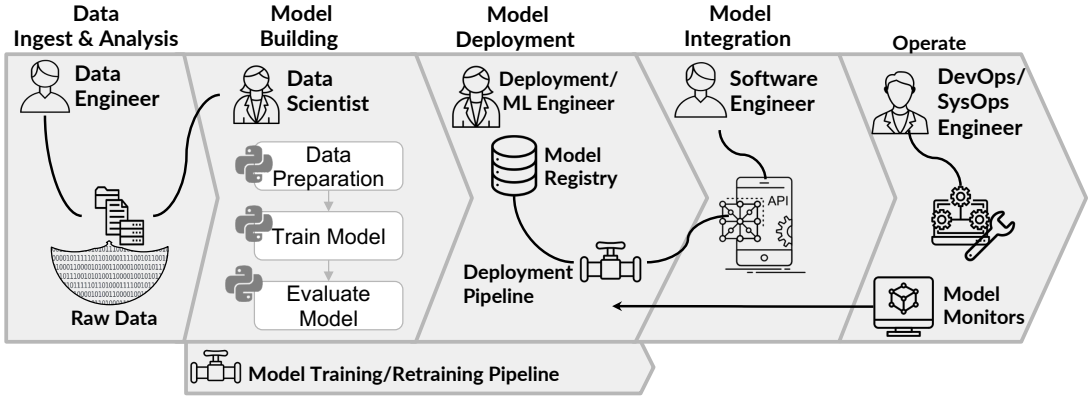
Example workflow with multiple hand offs:



Operationalizing Machine Learning

Accelerate the path to production

Automating the tasks within a workflow step

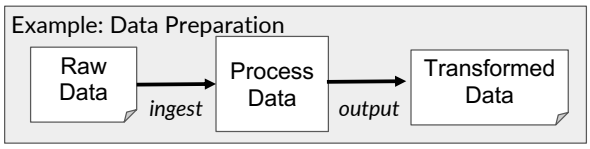


Operationalizing Machine Learning

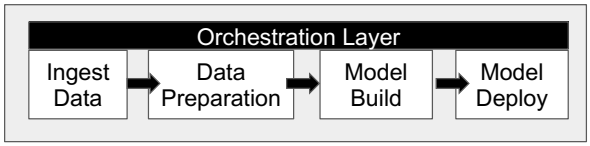
Accelerate the path to production

Automation vs Orchestration

Automation: Automate a **task** (Ex. Data Preparation) to perform a specific activity or produce defined artifacts based on the inputs or triggers of that task without human intervention



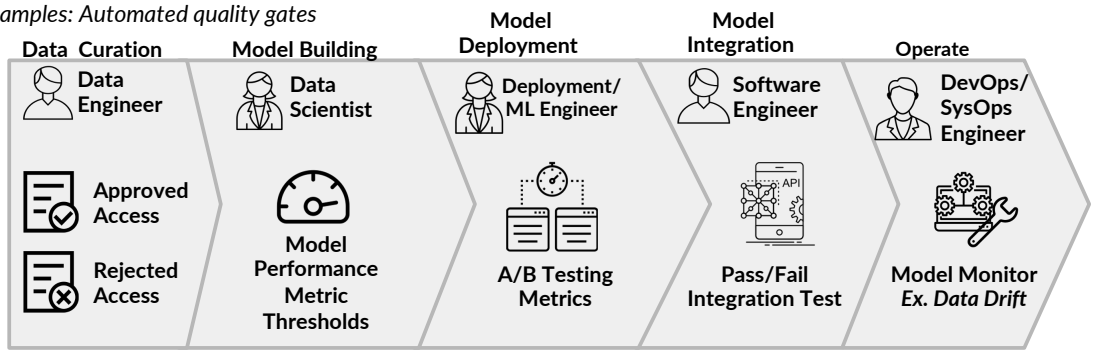
Orchestration: Orchestrate the steps of a workflow that contain a **collection of tasks**



Operationalizing Machine Learning

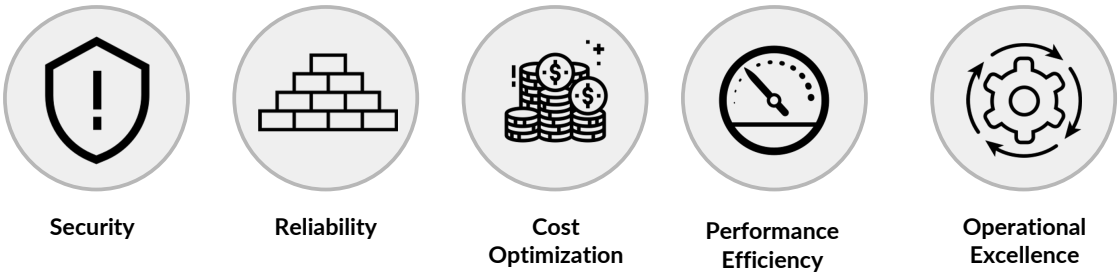
Improve the quality of deployed models

Examples: Automated quality gates



Operationalizing Machine Learning

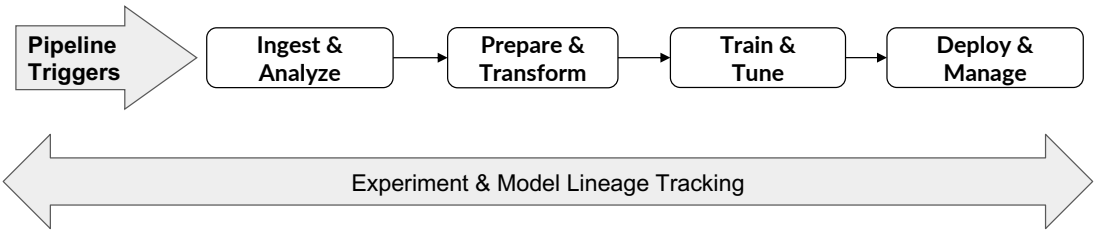
Key Considerations



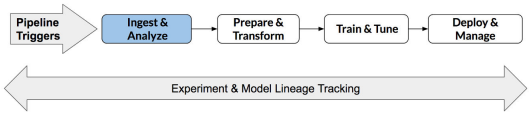
Creating Machine Learning Pipelines

Creating Machine Learning Pipelines

Building Effective Pipelines

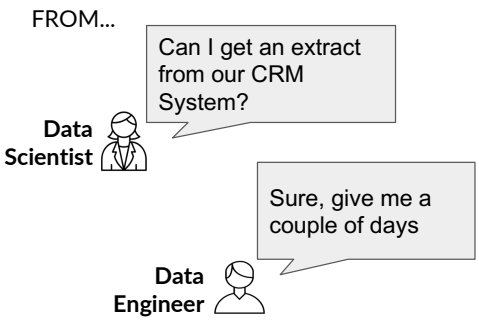


Data Tasks

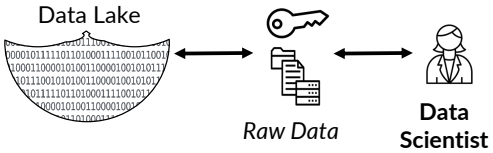


Data Ingestion for Model Development

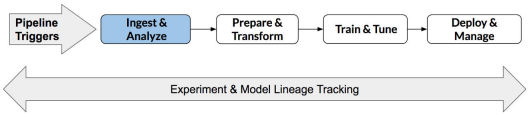
FROM...



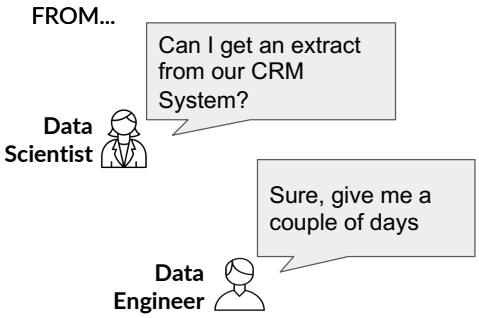
TO...

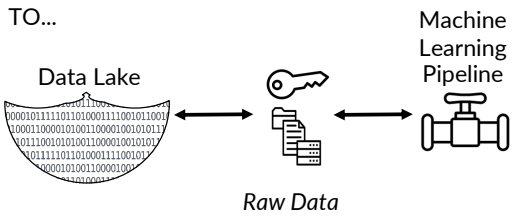


Data Tasks

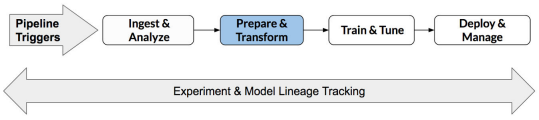


Data Ingestion for Model Retraining

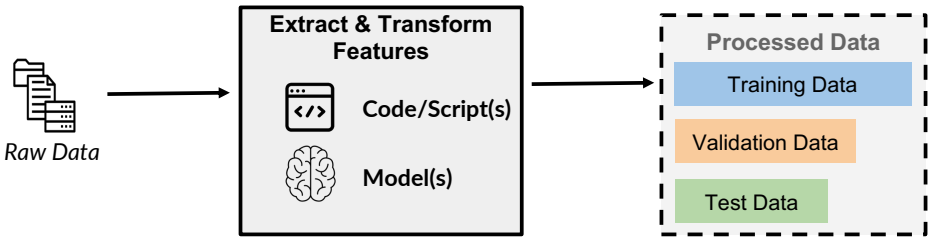




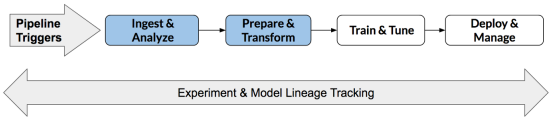
Data Tasks



Data Pre-Processing & Feature Engineering

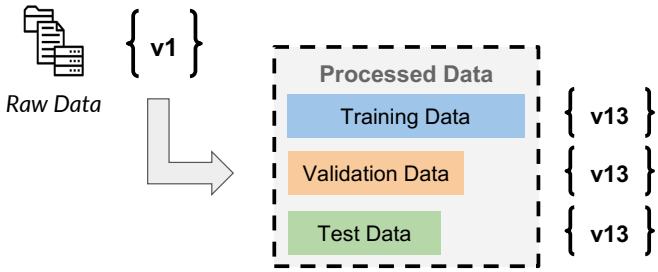


Data Tasks

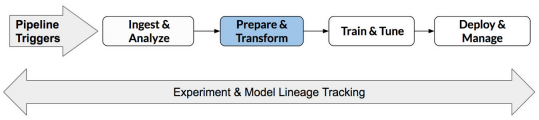


Data Versioning

Examples:

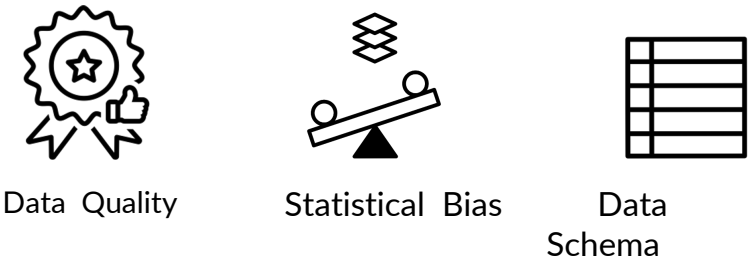


Data Tasks

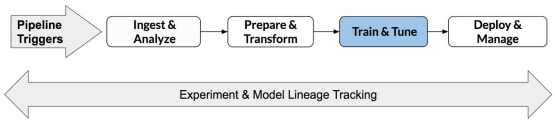


Data Validation

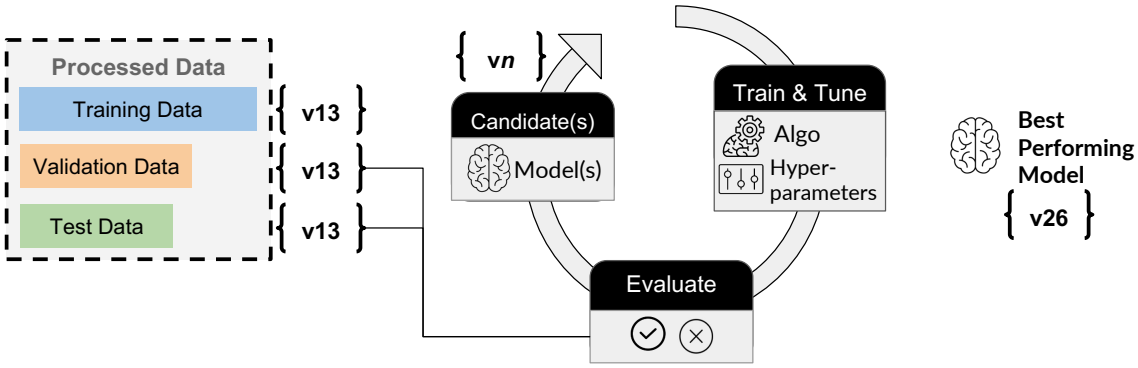
Examples:



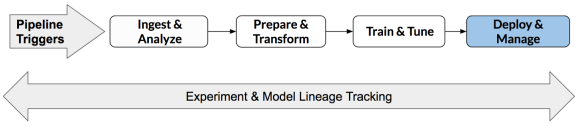
Model Building Tasks



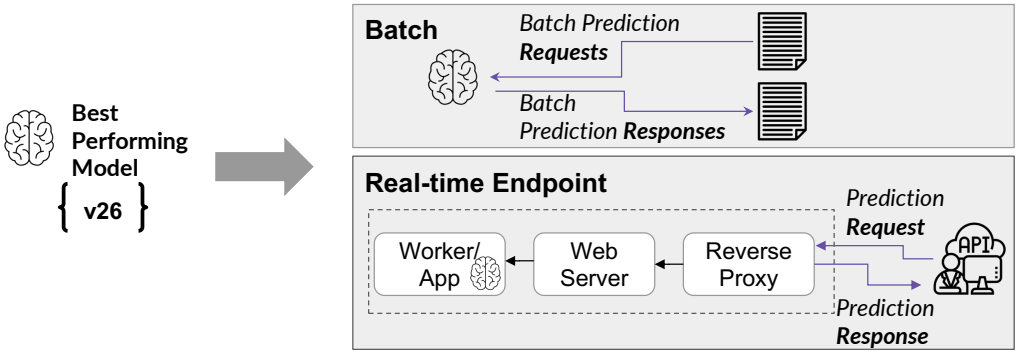
Model Training, Evaluation & Versioning



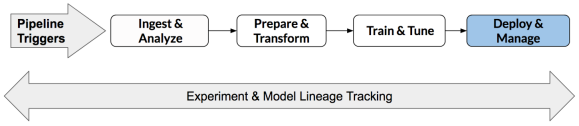
Model Deployment Tasks



Model Deployment & Consumption



Operating Tasks



Logging & Monitoring

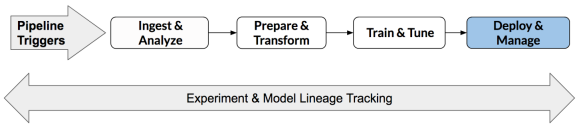
Logging →

- Model Data
- System Data

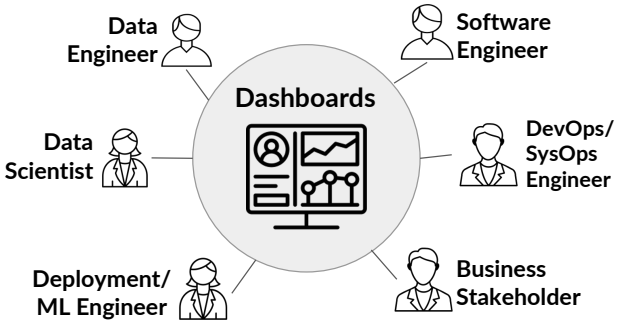
Monitoring →

- Collect Metrics
- Setup Alerts
- Trigger Automated Flows

Operating Tasks



Additional Feedback Mechanisms

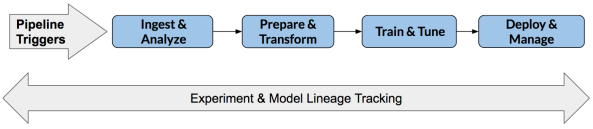


- Each persona can have different motivations and needs for monitors, logging and dashboards.
- Examples:
  - Pipeline Status
  - System Performance
  - Model Performance

Machine Learning Pipelines

Pipeline Orchestration: Bringing It Together

- ❑ Steps within **Task** can be automated
- ❑ Each set of tasks has **Inputs & Artifacts** produced as part of those steps
- ❑ **Orchestration** is required to coordinate the execution of tasks and steps within the tasks.

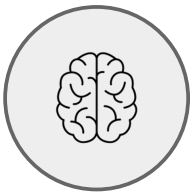


Model Lineage & Artifact Tracking

Model Lineage

What is Model Lineage?

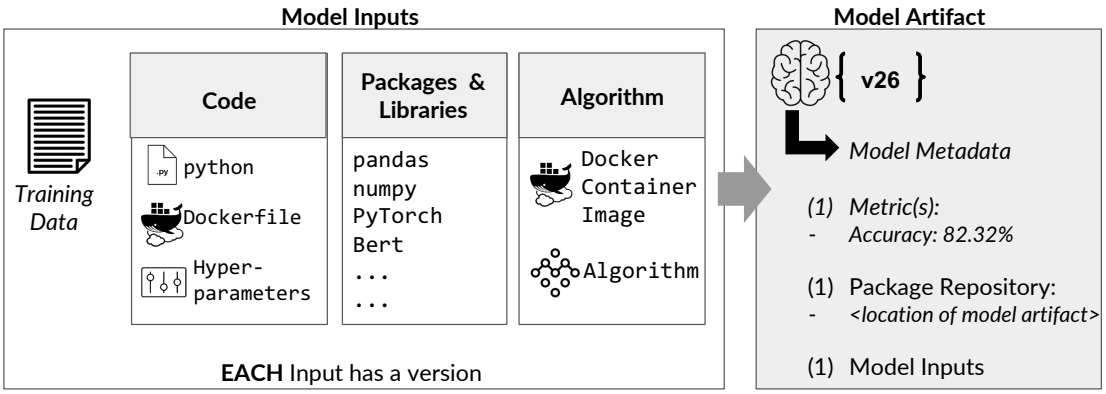
For **EACH** version of a trained model:



- ❑ Version(s) of data used
- ❑ Version(s) of code/hyperparameters used
- ❑ Version(s) of algorithm/framework
- ❑ Version(s) of training docker image
- ❑ Version(s) of packages/libraries

Model Lineage

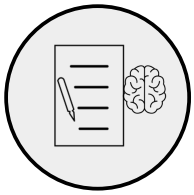
Model Lineage Example



Model Lineage

Model Registry

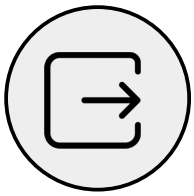
What is a Model Registry?



- ❑ Centrally manage model metadata and model artifacts
- ❑ Track which models are deploy across environments

Artifact Tracking

What is Artifact Tracking?

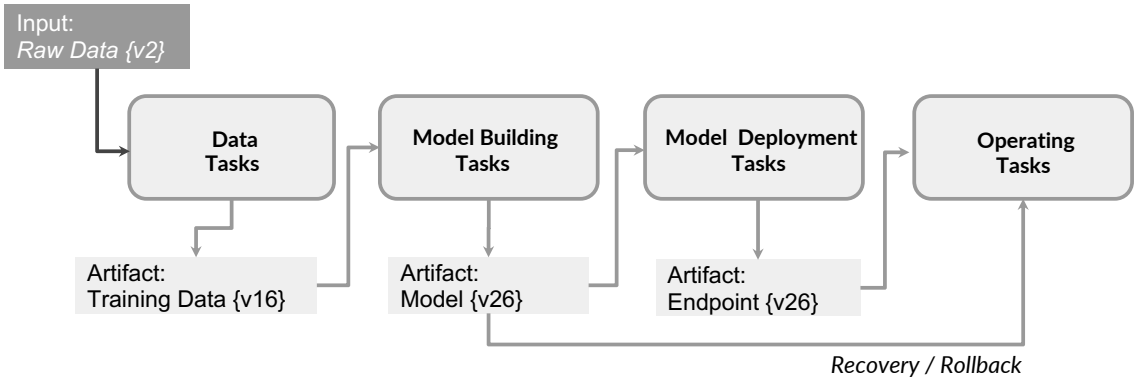


- An **Artifact** is the output of a step or task can be consumed by the next step in a pipeline or deployed directly for consumption

Artifact Tracking

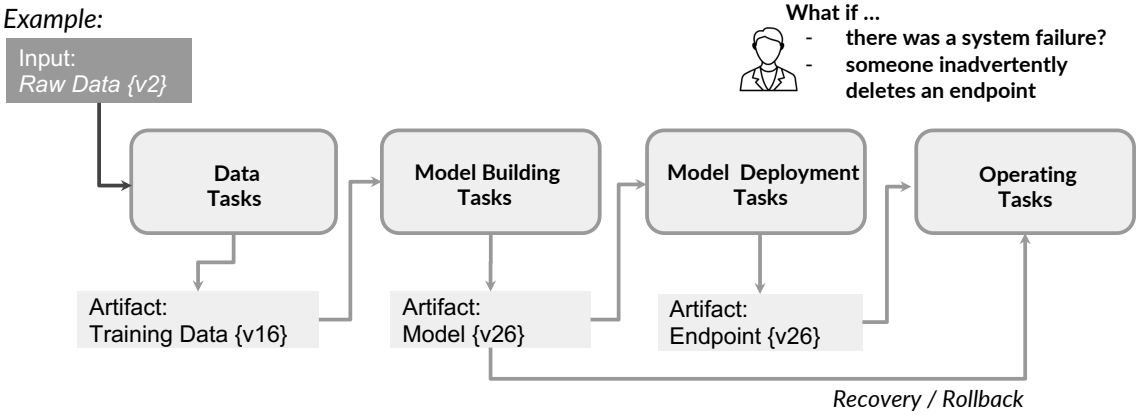
Pipeline Manifest

Example:



Artifact Tracking

Pipeline Manifest - Why it matters



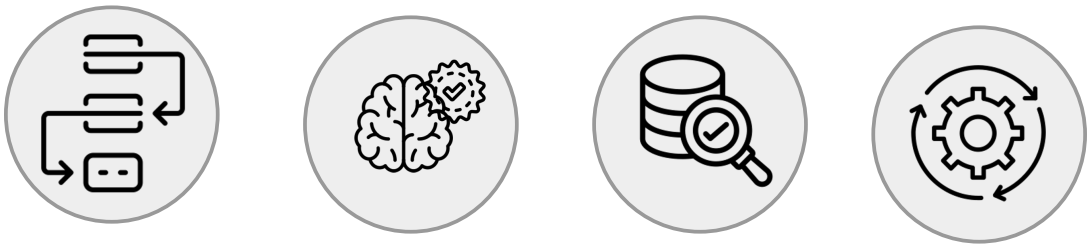
Machine Learning Pipelines

with
Amazon SageMaker Pipelines

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Amazon SageMaker Pipelines



Create & visualize automated workflows

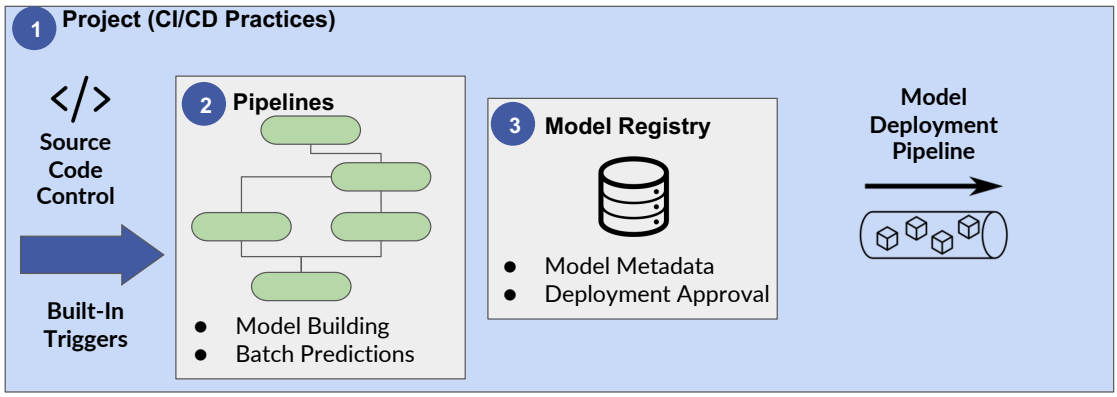
Choose the best performing model to deploy

Automatic tracking of models

Bring CI/CD to Machine Learning

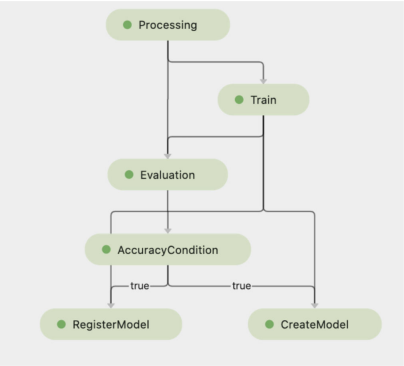
Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



SageMaker Pipelines

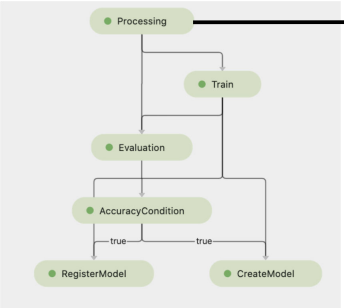
Pipelines



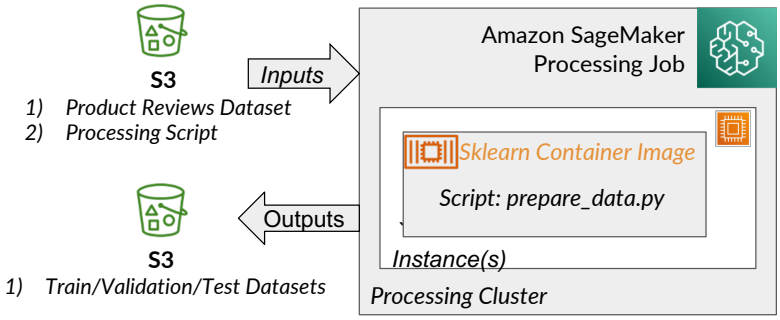
- ❑ Create Pipelines to build and evaluate models
- ❑ Python SDK for building workflows
- ❑ Pipeline visualization available through Amazon SageMaker Studio
- ❑ Fully managed pipelines - no servers to manage

SageMaker Pipelines

Processing Step

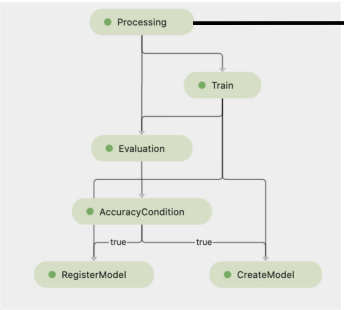


Use Amazon SageMaker Processing to process data set for training →



SageMaker Pipelines

Processing Step

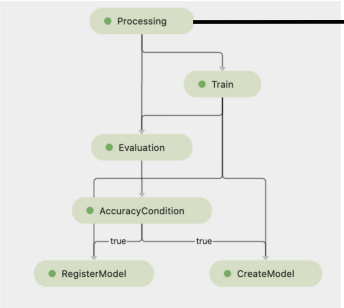


Define Step Inputs & Outputs →

```
processing_inputs = [
    ProcessingInput(
     input_name='customer-reviews-input-data',
     source='s3://...',
     destination='/opt/ml/processing/input/data/',
     s3_data_distribution_type='ShardedByS3Key'
    )
]

processing_outputs=[
         ProcessingOutput(...)
]
```


SageMaker Pipelines

Processing Step

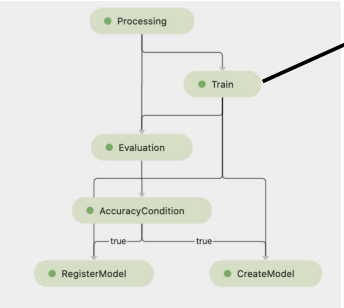


Configure the Processing Step →

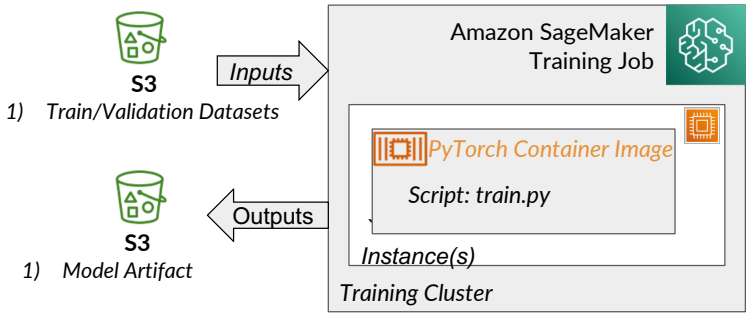
```
processing_step = ProcessingStep(
         name='Processing',
         code='src/prepare_data.py',
         processor=processor,
         inputs=processing_inputs,
         outputs=processing_outputs,
         job_arguments=[
         '--train-split-percentage',
         str(train_split_percentage.default_value,
         ...
     ]
)
```

SageMaker Pipelines

Training Step

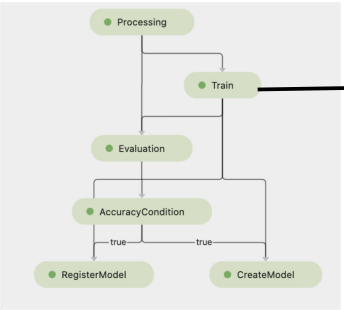


Use Amazon SageMaker Training Jobs to train the model using the outputs from the previous step as input →



SageMaker Pipelines

Training Step

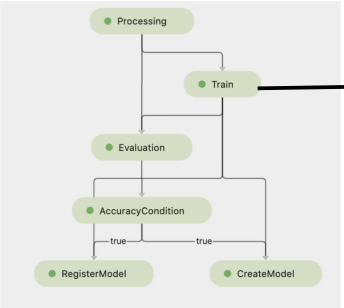


Configure Hyperparameters →

```
hyperparameters={
          'max_seq_length': max_seq_length,
          'epochs': epochs,
          'learning_rate': learning_rate,
     ...
}
```


SageMaker Pipelines

Training Step

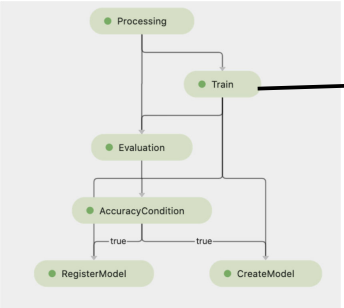


Configure Estimator →

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator
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,
          metric_definitions=metric_definitions,
          input_mode=input_mode
)
```

SageMaker Pipelines

Training Step



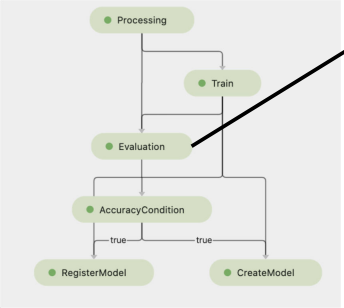
Configure the Training Step →

Output from Processing Step

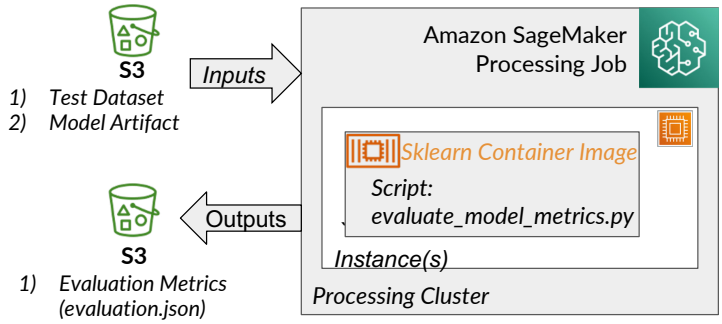
```
training_step = TrainingStep(
        name='Train',
        estimator=estimator,
        inputs={
        'train': TrainingInput(
        s3_data=processing_step.properties.ProcessingOu
tputConfig.Outputs[
                    'sentiment-train'
        ].S3Output.S3Uri,
        content_type='text/csv'
        ),
        'validation': TrainingInput(...
        )
    }
)
```


SageMaker Pipelines

Evaluation Step

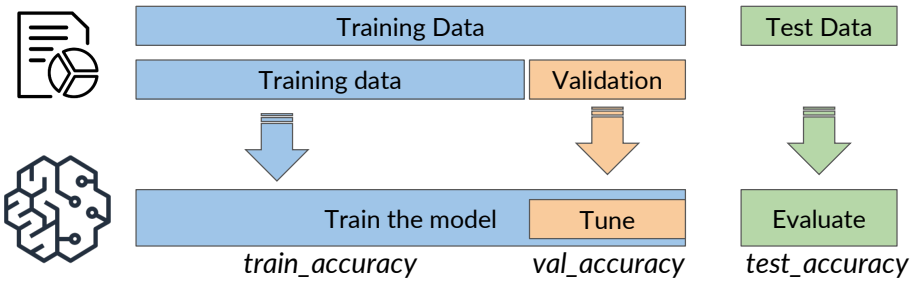


Use Amazon SageMaker Processing to evaluate trained model using test holdout dataset →



Model evaluation

- Evaluate the model with holdout test dataset



Code: *evaluate_model_metrics.py*

```
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
...

def predict_fn(input_data, model):
    model.eval()
    ...
    return predicted_classes_jsonlines

...
y_test = df_test_reviews['review_body'].map(predict)
y_actual = df_test_reviews['sentiment'].astype('int64')

print(classification_report(y_true=y_test, y_pred=y_actual))

accuracy = accuracy_score(y_true=y_test, y_pred=y_actual)
print('Test accuracy: ', accuracy)
```

Define model predict function

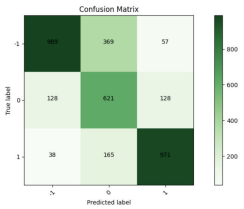
Use "test" holdout data

Calculate test accuracy

Analyze results

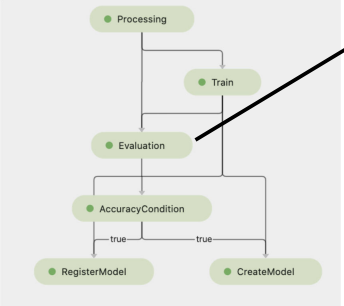
```
from pprint import pprint

evaluation_json = sagemaker.s3.S3Downloader.read_file(
     "{}/evaluation.json".format(evaluation_metrics_s3_uri))

pprint(json.loads(evaluation_json))
```

```
>> {'metrics': {'accuracy': {'value': 0.7458165031736872}}}
```



SageMaker Pipelines

Evaluation Step

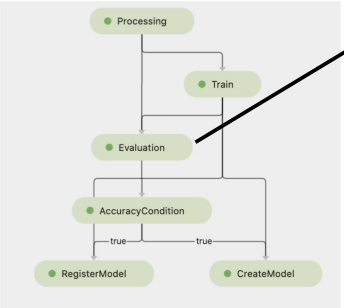


Define Output →

```
from sagemaker.workflow.properties import
PropertyFile

evaluation_report = PropertyFile(
          name='EvaluationReport',
          output_name='metrics',
          path='evaluation.json'
)
```

SageMaker Pipelines

Evaluation Step

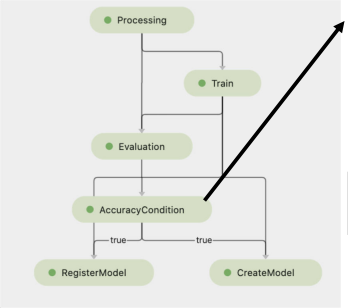


Configure Processing Step →

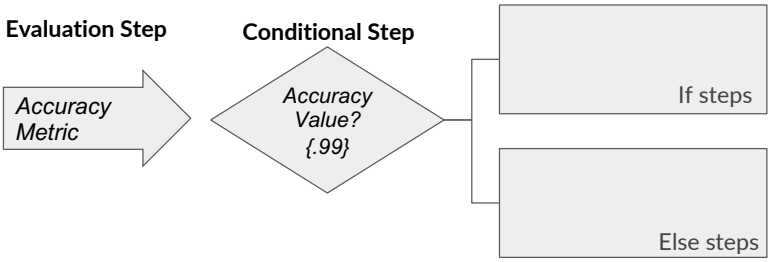
```
evaluation_step = ProcessingStep(
            name='EvaluateModel',
            processor=evaluation_processor,
            code='src/evaluate_model_metrics.py',
            inputs=[
            ProcessingInput(...),...
            ],
            outputs=[
            ProcessingOutput(...),
            ],
            job_arguments=[...],
            property_files=[evaluation_report],
)
```


SageMaker Pipelines

Condition Step

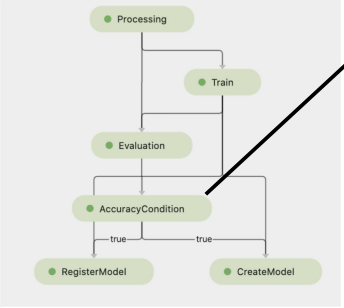


Use Amazon SageMaker Pipelines *Condition Step* to conditionally execute step(s) →



SageMaker Pipelines

Condition Step



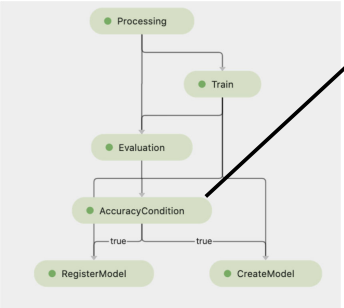
Define a Condition & Import Conditional Workflow Step →

```
min_accuracy_value = ParameterFloat(
            name="MinAccuracyValue",
            default_value=0.01
)
```

```
from sagemaker.workflow.conditions import
ConditionGreaterThanOrEqualTo

from sagemaker.workflow.condition_step import (
            ConditionStep,
            JsonGet,
)
```

SageMaker Pipelines

Condition Step

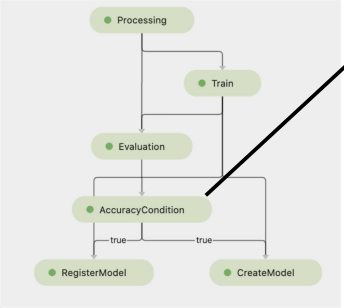


Configure the Condition →

```
minimum_accuracy_condition =
ConditionGreaterThanOrEqualTo(
          left=JsonGet(
          step=evaluation_step,
          property_file=evaluation_report,
          json_path="metrics.accuracy.value",
          ),
          right=min_accuracy_value # accuracy
)
```


SageMaker Pipelines

Condition Step

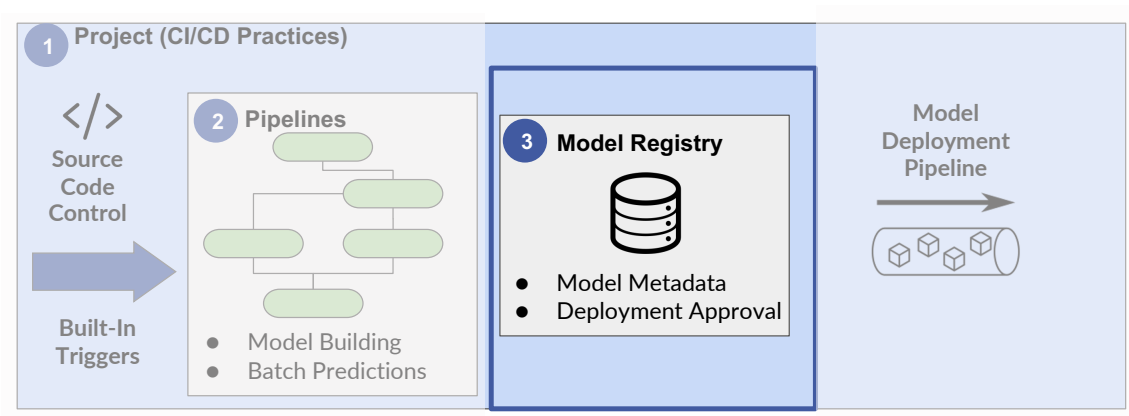


Configure the Condition Step →

```
minimum_accuracy_condition_step = ConditionStep(
          name="AccuracyCondition",
          conditions=[minimum_accuracy_condition],
     # success, continue with model registration
          if_steps=[register_step, create_step],
          else_steps=[], # fail, end the pipeline
)
```


Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



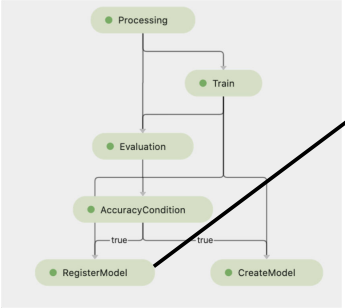
SageMaker Pipelines

Model Registry

- ❑ Catalog models for production
- ❑ Manage model versions & metadata
- ❑ Manage the approval status of a model
- ❑ Trigger model deployment pipeline

SageMaker Pipelines

Register Model Step

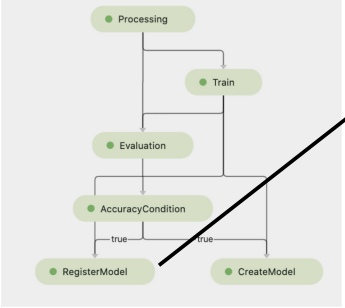


Define deployment image for inference →

```
inference_image_uri = sagemaker.image_uris.retrieve(
          framework="pytorch",
          region=region,
          version="1.6.0",
          py_version="py36",
          instance_type=deploy_instance_type,
          image_scope="inference"
)
```


SageMaker Pipelines

Register Model Step

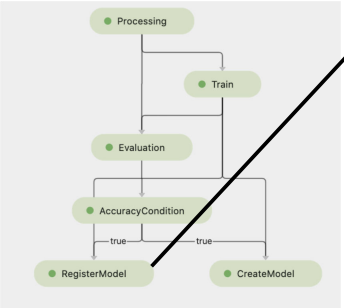


Define model metrics to be stored as metadata →

```
from sagemaker.model_metrics import MetricsSource,
ModelMetrics

model_metrics = ModelMetrics(
         model_statistics=MetricsSource(
         s3_uri="s3://..."),
         content_type="application/json"
         )
)
```

SageMaker Pipelines

Register Model Step

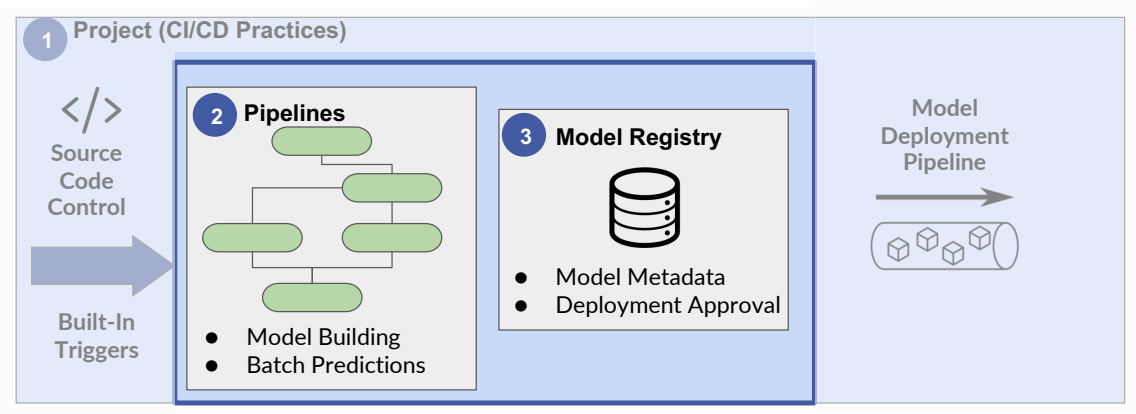


Configure the Register Model Step →

```
register_step = RegisterModel(
            name="RegisterModel",
            estimator=estimator,
            image_uri=...,
      model_data=
            training_step.properties.ModelArtifacts.S3ModelArtifacts,
      content_types=["application/jsonlines"],
      response_types=["application/jsonlines"],
      inference_instances=[deploy_instance_type],
      transform_instances=['ml.m5.xlarge'], # batch transform
      model_package_group_name=model_package_group_name,
      approval_status=model_approval_status,
      model_metrics=model_metrics)
```

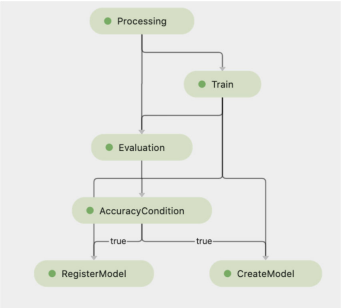

Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



SageMaker Pipelines

Bringing It All Together

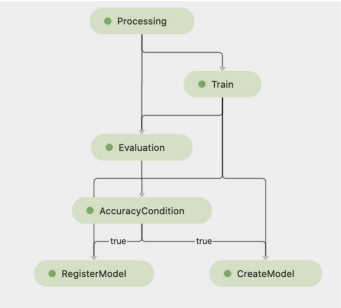


Configure the Pipeline →

```
from sagemaker.workflow.pipeline import Pipeline

pipeline = Pipeline(
      name=pipeline_name,
            parameters=[
            input_data,
            processing_instance_count,
            ...
      ],
      steps=[processing_step, training_step, evaluation_step,
      minimum_accuracy_condition_step], sagemaker_session=sess,
)
```

SageMaker Pipelines

Bringing It All Together



Create & Execute the Pipeline →

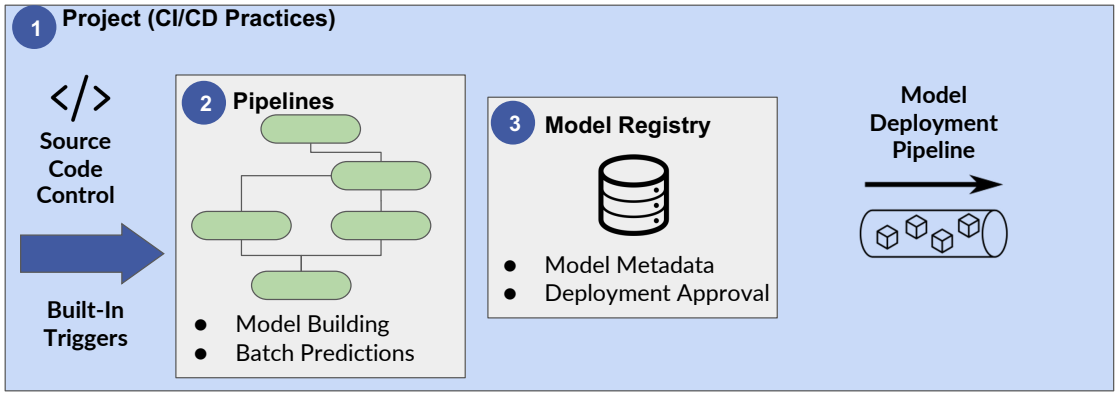
```
response = pipeline.create(role_arn=role)

pipeline_arn = response["PipelineArn"]

execution = pipeline.start(
            parameters=dict(
            InputData=raw_input_data_s3_uri,
            ProcessingInstanceCount=1,
      ...
            )
)
```


Machine Learning Pipelines

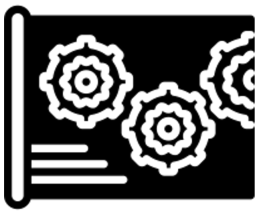
with
Amazon SageMaker Projects

Amazon SageMaker Pipelines

SageMaker Pipelines has 3 components ....



SageMaker Projects



- Create end-to-end ML solutions with CI/CD practices
- Incorporates source/version control
- 3 Built-In MLOps Project Templates covering:
  - Build, Train, Deploy
  - Build, Train
  - Deploy
- Ability to bring custom Project templates

Build a SageMaker Pipeline to train and deploy a BERT-Based text classifier

Introduction

In this lab, you will do the following:

- Define and run a pipeline using a directed acyclic graph (DAG) with specific pipeline parameters and model hyper-parameters
- Define a processing step that cleans, balances, transforms, and splits our dataset into train, validation, and test dataset
- Define a training step that trains a model using the train and validation datasets
- Define a processing step that evaluates the trained model's performance on the test dataset
- Define a register model step that creates a model package from the trained model
- Define a conditional step that checks the model's performance and conditionally registers the model for deployment

Table of Contents

- 1. Configure the dataset and processing step
  - 1.1. Configure S3 path for raw input data
  - 1.2. Configure processing step
    - Exercise 1
    - Exercise 2
- 2. Configure training step
  - 2.1. Define parameters
  - 2.2. Configure hyper-parameters
  - 2.3. Configure model-evaluation metrics
  - 2.4. Configure the `PyTorchEstimator`
  - 2.5. Setup pipeline step caching
  - 2.6. Configure the `TrainingStep`
    - Exercise 3
- 3. Configure model-evaluation step
- 4. Configure and register model step
  - 4.1. Configure the model for deployment
  - 4.2. Register the model for deployment
    - Exercise 4
- 5. Create model for deployment step
  - Exercise 5
  - Exercise 6

- 6. Check accuracy condition step
- 7. Create pipeline
  - 7.1. Define a pipeline of parameters, steps, and conditions
  - 7.2. Start Pipeline
  - 7.3. Wait for pipeline execution
  - 7.4. Describe completed pipeline
  - 7.5. Wait for the pipeline to complete
- 8. Evaluate the model
  - 8.1. Describe evaluation metrics
    - Exercise 7
  - 8.2. Review the evaluation report
  - 8.3. List pipeline artifacts
    - Exercise 8
- 9. Deploy and test the model
  - 9.1. Approve trained model
  - 9.2. Deploy model
  - 9.3. Create endpoint from registry
  - 9.4. Test model
  - 9.5. SageMaker Studio extensions

Terminology

This notebook focuses on the following features of Amazon SageMaker Pipelines:

- **Pipelines** - a directed acyclic graph (DAG) of steps and conditions to orchestrate SageMaker jobs and resource creation
- **Processing job steps** - a simplified, managed experience on SageMaker to run data processing workloads, such as feature engineering, data validation, model evaluation, and model explainability
- **Training job steps** - an iterative process that teaches a model to make predictions on new data by presenting examples from a training dataset
- **Conditional step execution** - provides conditional execution of branches in a pipeline
- **Registering models** - register a model in a model registry to create a deployable models in Amazon SageMaker
- **Parameterized pipeline executions** - allows pipeline executions to vary by supplied parameters
- **Model endpoint** - hosts the model as a REST endpoint to serve predictions from new data

BERT Pipeline

The pipeline that you will create follows a typical machine learning application pattern of pre-processing, training, evaluation, and model registration.

In the processing step, you will perform feature engineering to transform the `review_body` text into BERT embeddings using the pre-trained BERT model and split the dataset into train, validation and test files. The transformed dataset is stored in a feature store. To optimize for Tensorflow training, the transformed dataset files are saved using the TFRecord format in Amazon S3.

In the training step, you will fine-tune the BERT model to the customer reviews dataset and add a new classification layer to predict the `sentiment` for a given `review_body`.

In the evaluation step, you will take the trained model and a test dataset as input, and produce a JSON file containing classification evaluation metrics.

In the condition step, you will register the trained model if the accuracy of the model, as determined by our evaluation step, exceeds a given threshold value.

First, install the required modules.

In [1]:
```
# 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 [2]:
```
import os
import sagemaker
import logging
import boto3
import sagemaker
import pandas as pd
import json
import botocore
from botocore.exceptions import ClientError

config = botocore.config.Config(user_agent_extra='dlai-pds/c2/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
```

Setup the pipeline name.

In [3]:
```
import time
timestamp = int(time.time())

pipeline_name = 'BERT-pipeline-{}'.format(timestamp)
```

1. Configure the dataset and processing step

1.1. Configure S3 path for raw input data

The raw dataset is in the public S3 bucket. Let's start by specifying the S3 location of it:

In [4]:
```
raw_input_data_s3_uri = 's3://dlai-practical-data-science/data/raw/'
print(raw_input_data_s3_uri)
```

```
s3://dlai-practical-data-science/data/raw/
```

List the files in the S3 bucket (in this case it will be just one file):

In [5]:
```
!aws s3 ls $raw_input_data_s3_uri
```

```
2021-04-30 02:21:06    8457214 womens_clothing_ecommerce_reviews.csv
```

1.2. Configure processing step

For the pipeline workflow you will need to create workflow parameters of a specific type: integer, string, or float.

In [6]:
```
from sagemaker.workflow.parameters import (
    ParameterInteger,
    ParameterString,
    ParameterFloat,
)
```

Now set the parameters for the processing step.

In [7]:
```
processing_instance_type = ParameterString(
    name="ProcessingInstanceType",
    default_value="ml.c5.2xlarge"
)

processing_instance_count = ParameterInteger(
    name="ProcessingInstanceCount",
    default_value=1
)

train_split_percentage = ParameterFloat(
    name="TrainSplitPercentage",
    default_value=0.90,
)

validation_split_percentage = ParameterFloat(
    name="ValidationSplitPercentage",
    default_value=0.05,
)

test_split_percentage = ParameterFloat(
    name="TestSplitPercentage",
    default_value=0.05,
)

balance_dataset = ParameterString(
    name="BalanceDataset",
    default_value="True",
)

max_seq_length = ParameterInteger(
    name="MaxSeqLength",
    default_value=128,
)

feature_store_offline_prefix = ParameterString(
    name="FeatureStoreOfflinePrefix",
    default_value="reviews-feature-store-" + str(timestamp),
)

feature_group_name = ParameterString(
    name="FeatureGroupName",
    default_value="reviews-feature-group-" + str(timestamp)
)

input_data = ParameterString(
    name="InputData",
    default_value=raw_input_data_s3_uri,
)
```

Setting up scikit-learn-based processor, pass the SageMaker execution role, processing instance type and instance count.

In [8]:
```
from sagemaker.sklearn.processing import SKLearnProcessor

processor = SKLearnProcessor(
    framework_version='0.23-1',
    role=role,
    instance_type=processing_instance_type,
    instance_count=processing_instance_count,
    env={'AWS_DEFAULT_REGION': region},
)
```

Now you will use the processor instance to construct a `ProcessingStep`, along with the input and output channels and the code that will be executed when the pipeline invokes pipeline execution. This is very similar to a processor instance's `run` method, for those familiar with the existing Python SDK.

Note the `"sentiment-train"`, `"sentiment-validation"` and `"sentiment-test"` named channels specified in the output configuration for the processing job. Such step `Properties` can be used in subsequent steps and will resolve to their runtime values at execution. In particular, you will call out this usage defining the training step.

In [9]:
```
from sagemaker.processing import ProcessingInput, ProcessingOutput
from sagemaker.workflow.steps import ProcessingStep

processing_inputs=[
    ProcessingInput(
        input_name='raw-input-data',
        source=input_data,
        destination='/opt/ml/processing/input/data/',
        s3_data_distribution_type='ShardedByS3Key'
    )
]

processing_outputs=[
    ProcessingOutput(output_name='sentiment-train',
                     source='/opt/ml/processing/output/sentiment/train',
                     s3_upload_mode='EndOfJob'),
    ProcessingOutput(output_name='sentiment-validation',
                     source='/opt/ml/processing/output/sentiment/validati
                     s3_upload_mode='EndOfJob'),
    ProcessingOutput(output_name='sentiment-test',
                     source='/opt/ml/processing/output/sentiment/test',
                     s3_upload_mode='EndOfJob')
]

processing_step = ProcessingStep(
    name='Processing',
    code='src/prepare_data.py',
    processor=processor,
    inputs=processing_inputs,
    outputs=processing_outputs,
    job_arguments=['--train-split-percentage', str(train_split_percentage
                   '--validation-split-percentage', str(validation_split_
                   '--test-split-percentage', str(test_split_percentage.d
                   '--balance-dataset', str(balance_dataset.default_value
                   '--max-seq-length', str(max_seq_length.default_value),
                   '--feature-store-offline-prefix', str(feature_store_of
                   '--feature-group-name', str(feature_group_name.default
                  ]
)

print(processing_step)
```

```
ProcessingStep(name='Processing', step_type=<StepTypeEnum.PROCESSING: 'Pr
ocessing'>)
```

Now you can call out the properties of the processing job as an object using the command `processing_step.properties`. To print out and explore the attributes use `__dict__` method.

In [10]:
```
# print out the list of the processing job properties
print(json.dumps(
    processing_step.properties.__dict__,
    indent=4, sort_keys=True, default=str
))
```

```
{
    "AppSpecification": "<sagemaker.workflow.properties.Properties object
at 0x7f52abbc7390>",
    "AutoMLJobArn": "<sagemaker.workflow.properties.Properties object at
0x7f52abc83f10>",
    "CreationTime": "<sagemaker.workflow.properties.Properties object at
0x7f52abc83990>",
    "Environment": "<sagemaker.workflow.properties.Properties object at 0
x7f52abbc7490>",
    "ExitMessage": "<sagemaker.workflow.properties.Properties object at 0
x7f52abbc7810>",
    "ExperimentConfig": "<sagemaker.workflow.properties.Properties object
at 0x7f52abbc7690>",
    "FailureReason": "<sagemaker.workflow.properties.Properties object at
0x7f52abbc7850>",
    "LastModifiedTime": "<sagemaker.workflow.properties.Properties object
at 0x7f52abc83c50>",
    "MonitoringScheduleArn": "<sagemaker.workflow.properties.Properties o
bject at 0x7f52abc83090>",
    "NetworkConfig": "<sagemaker.workflow.properties.Properties object at
0x7f52abbc74d0>",
    "ProcessingEndTime": "<sagemaker.workflow.properties.Properties objec
t at 0x7f52abbc7890>",
    "ProcessingInputs": "<sagemaker.workflow.properties.PropertiesList ob
ject at 0x7f52abbc7050>",
    "ProcessingJobArn": "<sagemaker.workflow.properties.Properties object
at 0x7f52abbc7790>",
    "ProcessingJobName": "<sagemaker.workflow.properties.Properties objec
t at 0x7f52abbc7150>",
    "ProcessingJobStatus": "<sagemaker.workflow.properties.Properties obj
ect at 0x7f52abbc77d0>",
    "ProcessingOutputConfig": "<sagemaker.workflow.properties.Properties
object at 0x7f52abbc7090>",
    "ProcessingResources": "<sagemaker.workflow.properties.Properties obj
ect at 0x7f52abbc7190>",
    "ProcessingStartTime": "<sagemaker.workflow.properties.Properties obj
ect at 0x7f52abbc78d0>",
    "RoleArn": "<sagemaker.workflow.properties.Properties object at 0x7f5
2abbc7650>",
    "StoppingCondition": "<sagemaker.workflow.properties.Properties objec
t at 0x7f52abbc7310>",
    "TrainingJobArn": "<sagemaker.workflow.properties.Properties object a
t 0x7f52abc83f50>",
    "_path": "Steps.Processing",
    "_shape_name": "DescribeProcessingJobResponse"
}
```

Pull the channel `sentiment-train` from the output configuration of the processing job. Print out the attributes of the resulting object:

In [11]:
```
print(json.dumps(
    processing_step.properties.ProcessingOutputConfig.Outputs['sentiment-
    indent=4, sort_keys=True, default=str
))
```

```
{
    "AppManaged": "<sagemaker.workflow.properties.Properties object at 0x
7f52abc83550>",
    "FeatureStoreOutput": "<sagemaker.workflow.properties.Properties obje
ct at 0x7f52abc83210>",
    "OutputName": "<sagemaker.workflow.properties.Properties object at 0x
7f52abc83d10>",
    "S3Output": "<sagemaker.workflow.properties.Properties object at 0x7f
52abc83e10>",
    "_path": "Steps.Processing.ProcessingOutputConfig.Outputs['sentiment-
train']",
    "_shape_name": "ProcessingOutput"
}
```

Now you can pull and print out attributes of the S3 output path related to the `sentiment-train` output channel:

In [12]:
```
print(json.dumps(
    processing_step.properties.ProcessingOutputConfig.Outputs['sentiment-
    indent=4, sort_keys=True, default=str
))
```

```
{
    "__str__": "S3Uri",
    "_path": "Steps.Processing.ProcessingOutputConfig.Outputs['sentiment-
train'].S3Output.S3Uri",
    "_shape_name": "S3Uri"
}
```

Exercise 1

Pull and print out attributes of the S3 output path object related to the `sentiment-test` output channel.

Instructions: Use the example in the cell above.

In [13]:
```
print(json.dumps(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    processing_step.properties.ProcessingOutputConfig.Outputs['sentiment-
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    indent=4, sort_keys=True, default=str
))
```

```
{
    "__str__": "S3Uri",
    "_path": "Steps.Processing.ProcessingOutputConfig.Outputs['sentiment-
test'].S3Output.S3Uri",
    "_shape_name": "S3Uri"
}
```

These objects can be passed into the next steps of the workflow. Also, you can pull the arguments of the processing step with the corresponding function. The result is in the dictionary format. Review the keys of this dictionary:

In [14]:
```
processing_step.arguments.keys()
```

Out[14]:
```
dict_keys(['ProcessingResources', 'AppSpecification', 'RoleArn', 'Process
ingInputs', 'ProcessingOutputConfig', 'Environment'])
```

Pull and review processing inputs from the arguments of the processing step:

In [15]:
```
processing_step.arguments['ProcessingInputs']
```

Out[15]:
```
[{'InputName': 'raw-input-data',
  'AppManaged': False,
  'S3Input': {'S3Uri': ParameterString(name='InputData', parameter_type=<
ParameterTypeEnum.STRING: 'String'>, default_value='s3://dlai-practical-d
ata-science/data/raw/'),
   'LocalPath': '/opt/ml/processing/input/data/',
   'S3DataType': 'S3Prefix',
   'S3InputMode': 'File',
   'S3DataDistributionType': 'ShardedByS3Key',
   'S3CompressionType': 'None'}},
 {'InputName': 'code',
  'AppManaged': False,
  'S3Input': {'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker-s
cikit-learn-2022-09-04-06-59-33-899/input/code/prepare_data.py',
   'LocalPath': '/opt/ml/processing/input/code',
   'S3DataType': 'S3Prefix',
   'S3InputMode': 'File',
   'S3DataDistributionType': 'FullyReplicated',
   'S3CompressionType': 'None'}}]
```

Exercise 2

Pull and review configuration of the processing outputs from the arguments of the processing step.

Instructions: Find the required key in the `arguments` dictionary and pull the corresponding value following the example above.

In [16]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
processing_step.arguments['ProcessingOutputConfig'] # Replace None
### END SOLUTION - DO NOT delete this comment for grading purposes
```

Out[16]:
```
{'Outputs': [{'OutputName': 'sentiment-train',
   'AppManaged': False,
   'S3Output': {'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker
-scikit-learn-2022-09-04-06-59-33-489/output/sentiment-train',
    'LocalPath': '/opt/ml/processing/output/sentiment/train',
    'S3UploadMode': 'EndOfJob'}},
  {'OutputName': 'sentiment-validation',
   'AppManaged': False,
   'S3Output': {'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker
-scikit-learn-2022-09-04-06-59-33-489/output/sentiment-validation',
    'LocalPath': '/opt/ml/processing/output/sentiment/validation',
    'S3UploadMode': 'EndOfJob'}},
  {'OutputName': 'sentiment-test',
   'AppManaged': False,
   'S3Output': {'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker
-scikit-learn-2022-09-04-06-59-33-489/output/sentiment-test',
    'LocalPath': '/opt/ml/processing/output/sentiment/test',
    'S3UploadMode': 'EndOfJob'}}]}
```

2. Configure training step

2.1. Define parameters

Setup the parameters for the workflow.

In [17]:
```
freeze_bert_layer = ParameterString(
    name="FreezeBertLayer",
    default_value="False",
)

epochs = ParameterInteger(
    name="Epochs",
    default_value=3
)

learning_rate = ParameterFloat(
    name="LearningRate",
    default_value=0.00001
)

train_batch_size = ParameterInteger(
    name="TrainBatchSize",
    default_value=64
)

train_steps_per_epoch = ParameterInteger(
    name="TrainStepsPerEpoch",
    default_value=50
)

validation_batch_size = ParameterInteger(
    name="ValidationBatchSize",
    default_value=64
)

validation_steps_per_epoch = ParameterInteger(
    name="ValidationStepsPerEpoch",
    default_value=50
)

seed = ParameterInteger(
    name="Seed",
    default_value=42
)

run_validation = ParameterString(
    name="RunValidation",
    default_value="True",
)

train_instance_count = ParameterInteger(
    name="TrainInstanceCount",
    default_value=1
```

```
)

train_instance_type = ParameterString(
    name="TrainInstanceType",
    default_value="ml.c5.9xlarge"
)

train_volume_size = ParameterInteger(
    name="TrainVolumeSize",
    default_value=256
)

input_mode = ParameterString(
    name="InputMode",
    default_value="File",
)
```

2.2. Configure hyper-parameters

Setup the dictionary that will be passed into the hyperparameters argument.

In [18]:
```
hyperparameters={
    'max_seq_length': max_seq_length,
    'freeze_bert_layer': freeze_bert_layer,
    'epochs': epochs,
    'learning_rate': learning_rate,
    'train_batch_size': train_batch_size,
    '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
}
```

2.3. Configure model-evaluation metrics

Choose loss and accuracy as the evaluation metrics.

In [19]:
```
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: 0.55 - val_acc: 74.64%
```

...will produce the following metrics in CloudWatch:

`validation:loss` = 0.55

`validation:accuracy` = 74.64

2.4. Configure the `PyTorchEstimator`

Configure an estimator and the input dataset. A typical training script loads data from the input channels, configures training with hyperparameters, trains a model, and saves a model to `model_dir` so that it can be hosted later.

In [20]:
```
from sagemaker.pytorch import PyTorch as PyTorchEstimator

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,
    metric_definitions=metric_definitions,
    input_mode=input_mode
)
```

2.5. Setup pipeline step caching

Step signature caching allows SageMaker Pipelines, before executing a step, to find a previous execution of a step that was called using the same arguments. Cache hit gets created if the previous execution is found. Then during execution instead of recomputing the step, pipelines propagates the values from the cache hit.

Timeout period is defined using ISO 8601 format, it can contain a year, month, week, day, hour, and minute value.

More details on SageMaker Pipeline step caching can be found here.

In [21]:
```
from sagemaker.workflow.steps import CacheConfig

cache_config = CacheConfig(enable_caching=True, expire_after="PT1H") # PT
```

2.6. Configure the `TrainingStep`

Now configure the `TrainingStep` calling the outputs of the processing step:

In [22]:
```
from sagemaker.inputs import TrainingInput
from sagemaker.workflow.steps import TrainingStep

training_step = TrainingStep(
    name='Train',
    estimator=estimator,
    inputs={
        'train': TrainingInput(
            s3_data=processing_step.properties.ProcessingOutputConfig.Out
                'sentiment-train'
            ].S3Output.S3Uri,
            content_type='text/csv'
        ),
        'validation': TrainingInput(
            s3_data=processing_step.properties.ProcessingOutputConfig.Out
                'sentiment-validation'
            ].S3Output.S3Uri,
            content_type='text/csv'
        )
    },
    cache_config=cache_config
)

print(training_step)
```

```
TrainingStep(name='Train', step_type=<StepTypeEnum.TRAINING: 'Training'>)
```

Exercise 3

Use `__dict__` method to print out attributes of the training step properties. Briefly review the result. The attributes match the object model of the DescribeTrainingJob response object.

In [23]:
```
### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
training_step.properties.__dict__ # Replace all None
### END SOLUTION - DO NOT delete this comment for grading purposes
```

Out[23]:
```
{'_path': 'Steps.Train',
 '_shape_name': 'DescribeTrainingJobResponse',
 'TrainingJobName': <sagemaker.workflow.properties.Properties at 0x7f52ab
de1790>,
 'TrainingJobArn': <sagemaker.workflow.properties.Properties at 0x7f52abd
e1d10>,
 'TuningJobArn': <sagemaker.workflow.properties.Properties at 0x7f52abde1
bd0>,
 'LabelingJobArn': <sagemaker.workflow.properties.Properties at 0x7f52abd
e1b50>,
 'AutoMLJobArn': <sagemaker.workflow.properties.Properties at 0x7f52abde1
550>,
 'ModelArtifacts': <sagemaker.workflow.properties.Properties at 0x7f52abd
e1c50>,
 'TrainingJobStatus': <sagemaker.workflow.properties.Properties at 0x7f52
abaa9ed0>,
 'SecondaryStatus': <sagemaker.workflow.properties.Properties at 0x7f52ab
aa9fd0>,
 'FailureReason': <sagemaker.workflow.properties.Properties at 0x7f52ac16
1690>,
```

```
 'HyperParameters': <sagemaker.workflow.properties.Properties at 0x7f52ab
ba8650>,
 'AlgorithmSpecification': <sagemaker.workflow.properties.Properties at 0
x7f52abba80d0>,
 'RoleArn': <sagemaker.workflow.properties.Properties at 0x7f52abba8d50>,
 'InputDataConfig': <sagemaker.workflow.properties.PropertiesList at 0x7f
52abba8a90>,
 'OutputDataConfig': <sagemaker.workflow.properties.Properties at 0x7f52a
bba8810>,
 'ResourceConfig': <sagemaker.workflow.properties.Properties at 0x7f52abb
a8890>,
 'VpcConfig': <sagemaker.workflow.properties.Properties at 0x7f52abba8e10
>,
 'StoppingCondition': <sagemaker.workflow.properties.Properties at 0x7f52
abba8350>,
 'CreationTime': <sagemaker.workflow.properties.Properties at 0x7f52abba8
4d0>,
 'TrainingStartTime': <sagemaker.workflow.properties.Properties at 0x7f52
abba8410>,
 'TrainingEndTime': <sagemaker.workflow.properties.Properties at 0x7f52ab
d86410>,
 'LastModifiedTime': <sagemaker.workflow.properties.Properties at 0x7f52a
bd86590>,
 'SecondaryStatusTransitions': <sagemaker.workflow.properties.PropertiesL
ist at 0x7f52abd863d0>,
 'FinalMetricDataList': <sagemaker.workflow.properties.PropertiesList at
0x7f52abd86250>,
 'EnableNetworkIsolation': <sagemaker.workflow.properties.Properties at 0
x7f52abd865d0>,
 'EnableInterContainerTrafficEncryption': <sagemaker.workflow.properties.
Properties at 0x7f52abd86150>,
 'EnableManagedSpotTraining': <sagemaker.workflow.properties.Properties a
t 0x7f52abd861d0>,
 'CheckpointConfig': <sagemaker.workflow.properties.Properties at 0x7f52a
bd86190>,
 'TrainingTimeInSeconds': <sagemaker.workflow.properties.Properties at 0x
7f52abd86350>,
 'BillableTimeInSeconds': <sagemaker.workflow.properties.Properties at 0x
7f52abd86310>,
 'DebugHookConfig': <sagemaker.workflow.properties.Properties at 0x7f52ab
d86550>,
 'ExperimentConfig': <sagemaker.workflow.properties.Properties at 0x7f52a
bd86490>,
 'DebugRuleConfigurations': <sagemaker.workflow.properties.PropertiesList
at 0x7f52aba96450>,
 'TensorBoardOutputConfig': <sagemaker.workflow.properties.Properties at
0x7f52aba96610>,
 'DebugRuleEvaluationStatuses': <sagemaker.workflow.properties.Properties
List at 0x7f52aba96990>,
 'ProfilerConfig': <sagemaker.workflow.properties.Properties at 0x7f52aba
96910>,
 'ProfilerRuleConfigurations': <sagemaker.workflow.properties.PropertiesL
ist at 0x7f52aba96290>,
 'ProfilerRuleEvaluationStatuses': <sagemaker.workflow.properties.Propert
iesList at 0x7f52aba96b10>,
 'ProfilingStatus': <sagemaker.workflow.properties.Properties at 0x7f52ab
a96a50>,
 'RetryStrategy': <sagemaker.workflow.properties.Properties at 0x7f52aba9
6590>,
 'Environment': <sagemaker.workflow.properties.Properties at 0x7f52aba962
50>}
```

3. Configure model-evaluation step

First, develop an evaluation script that will be specified in the model evaluation processing step. The evaluation script users the trained model and the test dataset to produce a JSON file with classification evaluation metrics such as accuracy.

After pipeline execution, you will examine the resulting `evaluation.json` for analysis.

The evaluation script performs the following steps:

- loads in the model
- reads in the test data
- issues a bunch of predictions against the test data
- builds a classification report, including accuracy
- saves the evaluation report to the evaluation directory

Create an instance of the `SKLearnProcessor` to run our evaluation script as a scikit-learn-based SageMaker processing job.

In [24]:
```
from sagemaker.sklearn.processing import SKLearnProcessor

evaluation_processor = SKLearnProcessor(
    framework_version='0.23-1',
    role=role,
    instance_type=processing_instance_type,
    instance_count=processing_instance_count,
    env={'AWS_DEFAULT_REGION': region},
    max_runtime_in_seconds=7200
)
```

Setup the output `PropertyFile`.

In [25]:
```
from sagemaker.workflow.properties import PropertyFile

evaluation_report = PropertyFile(
    name='EvaluationReport',
    output_name='metrics',
    path='evaluation.json'
)
```

Use the processor instance to construct a `ProcessingStep`, along with the input and output channels and the code that will be executed when the pipeline invokes pipeline execution. This is very similar to a processor instance's `run` method.

In [26]:
```
from sagemaker.processing import ProcessingInput, ProcessingOutput

evaluation_step = ProcessingStep(
    name='EvaluateModel',
    processor=evaluation_processor,
    code='src/evaluate_model_metrics.py',
    inputs=[
        ProcessingInput(
            source=training_step.properties.ModelArtifacts.S3ModelArtifac
            destination='/opt/ml/processing/input/model'
        ),
        ProcessingInput(
            source=processing_step.properties.ProcessingOutputConfig.Outp
            destination='/opt/ml/processing/input/data'
        )
    ],
    outputs=[
        ProcessingOutput(output_name='metrics',
                         s3_upload_mode='EndOfJob',
                         source='/opt/ml/processing/output/metrics/'),
    ],
    job_arguments=[
        '--max-seq-length', str(max_seq_length.default_value),
    ],
    property_files=[evaluation_report],
)
```

4. Configure and register model step

4.1. Configure the model for deployment

Use the estimator instance that was used for the training step to construct an instance of `RegisterModel`. The result of executing `RegisterModel` in a pipeline is a model package. A model package is a reusable model artifacts abstraction that packages all ingredients necessary for inference. Primarily, it consists of an inference specification that defines the inference image to use along with an optional model weights location.

A model package group is a collection of model packages. You can create a model package group for a specific ML business problem, and you can keep adding versions/model packages into it. Typically, customers are expected to create a ModelPackageGroup for a SageMaker workflow pipeline so that they can keep adding versions/model packages to the group for every workflow pipeline run.

The construction of `RegisterModel` is very similar to an estimator instance's `register` method, for those familiar with the existing Python SDK.

In particular, you will pass in the `S3ModelArtifacts` from the `training_step` properties.

Of note, here you will be provided a specific model package group name which will be used in the Model Registry and Continuous Integration/Continuous Deployment (CI/CD) work later on. Let's setup the variables.

In [27]:
```
model_approval_status = ParameterString(
    name="ModelApprovalStatus",
    default_value="PendingManualApproval"
)

deploy_instance_type = ParameterString(
    name="DeployInstanceType",
    default_value="ml.m5.large"
)

deploy_instance_count = ParameterInteger(
    name="DeployInstanceCount",
    default_value=1
)
```

In [28]:
```
model_package_group_name = f"BERT-Reviews-{timestamp}"

print(model_package_group_name)
```

```
BERT-Reviews-1662274772
```

Configure the `ModelMetrics` to be stored as metadata.

In [29]:
```
from sagemaker.model_metrics import MetricsSource, ModelMetrics

model_metrics = ModelMetrics(
    model_statistics=MetricsSource(
        s3_uri="{}/evaluation.json".format(
            evaluation_step.arguments["ProcessingOutputConfig"]["Outputs"
        ),
        content_type="application/json"
    )
)

print(model_metrics)
```

```
<sagemaker.model_metrics.ModelMetrics object at 0x7f52ab87b490>
```

Define deployment image for inference.

In [30]:
```
inference_image_uri = sagemaker.image_uris.retrieve(
    framework="pytorch",
    region=region,
    version="1.6.0",
    py_version="py36",
    instance_type=deploy_instance_type,
    image_scope="inference"
)
print(inference_image_uri)
```

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:1.6.0-cpu-
py36
```

4.2. Register the model for deployment

Exercise 4

Configure the register model step.

Instructions: Pass the inference image defined above into the `image_uri` argument of the function `RegisterModel`. Review the rest of the arguments.

In [31]:
```
from sagemaker.workflow.step_collections import RegisterModel

register_step = RegisterModel(
    name="RegisterModel",
    estimator=estimator,
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    image_uri=inference_image_uri, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    model_data=training_step.properties.ModelArtifacts.S3ModelArtifacts,
    content_types=["application/jsonlines"],
    response_types=["application/jsonlines"],
    inference_instances=[deploy_instance_type],
    transform_instances=[deploy_instance_type], # batch transform is not
    model_package_group_name=model_package_group_name,
    approval_status=model_approval_status,
    model_metrics=model_metrics
)
```

5. Create model for deployment step

Exercise 5

Configure model for deployment.

Instructions: Pass the same inference image into the `image_uri` argument of the function `Model`.

In [32]:
```
from sagemaker.model import Model

model_name = 'bert-model-{}'.format(timestamp)

model = Model(
    name=model_name,
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    image_uri=inference_image_uri, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    model_data=training_step.properties.ModelArtifacts.S3ModelArtifacts,
    sagemaker_session=sess,
    role=role,
)
```

Now configure create model input:

In [33]:
```
from sagemaker.inputs import CreateModelInput

create_inputs = CreateModelInput(
    instance_type=deploy_instance_type,
)
```

Exercise 6

Configure create model step for the workflow.

Instructions: Pass defined above model (the model object, not its name) and model inputs configuration into the related arguments of the function `CreateModelStep`.

In [34]:
```
from sagemaker.workflow.steps import CreateModelStep

create_step = CreateModelStep(
    name="CreateModel",
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    model=model, # Replace None
    inputs=create_inputs, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
)
```

6. Check accuracy condition step

Finally, you would like to only register this model if the accuracy of the model, as determined by our evaluation step `evaluation_step`, exceeded some value. A `ConditionStep` allows for pipelines to support conditional execution in the pipeline DAG based on conditions of step properties.

Below, you will:

- define a minimum accuracy value as a parameter
- define a `ConditionGreaterThan` on the accuracy value found in the output of the evaluation step, `evaluation_step`.
- use the condition in the list of conditions in a `ConditionStep`
- pass the `RegisterModel` step collection into the `if_steps` of the `ConditionStep`

In [35]:
```
min_accuracy_value = ParameterFloat(
    name="MinAccuracyValue",
    default_value=0.33 # random choice from three classes
)
```

In [36]:
```
from sagemaker.workflow.conditions import ConditionGreaterThanOrEqualTo
from sagemaker.workflow.condition_step import (
    ConditionStep,
    JsonGet,
)

minimum_accuracy_condition = ConditionGreaterThanOrEqualTo(
    left=JsonGet(
        step=evaluation_step,
        property_file=evaluation_report,
        json_path="metrics.accuracy.value",
    ),
    right=min_accuracy_value # minimum accuracy threshold
)

minimum_accuracy_condition_step = ConditionStep(
    name="AccuracyCondition",
    conditions=[minimum_accuracy_condition],
    if_steps=[register_step, create_step], # successfully exceeded or equ
    else_steps=[], # did not exceed the minimum accuracy, the model will
)
```

7. Create pipeline

7.1. Define a pipeline of parameters, steps, and conditions

Let's tie it all up into a workflow pipeline so you can execute it, and even schedule it.

A pipeline requires a `name`, `parameters`, and `steps`. Names must be unique within an `(account, region)` pair so you can append the timestamp to the name to reduce the chance of name conflict.

Note:

- All the parameters used in the definitions must be present.
- Steps passed into the pipeline need not be in the order of execution. The SageMaker workflow service will resolve the *data dependency* DAG as steps the execution complete.
- Steps must be unique to either pipeline step list or a single condition step if/else list.

In [37]:
```
from sagemaker.workflow.pipeline import Pipeline

pipeline = Pipeline(
    name=pipeline_name,
    parameters=[
        input_data,
        processing_instance_count,
        processing_instance_type,
        max_seq_length,
        balance_dataset,
        train_split_percentage,
        validation_split_percentage,
        test_split_percentage,
        feature_store_offline_prefix,
        feature_group_name,
        epochs,
        learning_rate,
        train_batch_size,
        train_steps_per_epoch,
        validation_batch_size,
        validation_steps_per_epoch,
        freeze_bert_layer,
        seed,
        train_instance_count,
        train_instance_type,
        train_volume_size,
        input_mode,
        run_validation,
        min_accuracy_value,
        model_approval_status,
        deploy_instance_type,
        deploy_instance_count
    ],
    steps=[processing_step, training_step, evaluation_step, minimum_accur
    sagemaker_session=sess,
)
```

Let's examine the JSON of the pipeline definition that meets the SageMaker Workflow Pipeline DSL specification.

By examining the definition, you are also confirming that the pipeline was well-defined, and that the parameters and step properties resolve correctly.

In [38]:
```
import json
from pprint import pprint

definition = json.loads(pipeline.definition())

pprint(definition)
```

```
No finished training job found associated with this estimator. Please mak
e sure this estimator is only used for building workflow config
{'Metadata': {},
 'Parameters': [{'DefaultValue': 's3://dlai-practical-data-science/data/r
aw/',
                 'Name': 'InputData',
                 'Type': 'String'},
                {'DefaultValue': 1,
```

```
                     'Name': 'ProcessingInstanceCount',
                     'Type': 'Integer'},
                    {'DefaultValue': 'ml.c5.2xlarge',
                     'Name': 'ProcessingInstanceType',
                     'Type': 'String'},
                    {'DefaultValue': 128,
                     'Name': 'MaxSeqLength',
                     'Type': 'Integer'},
                    {'DefaultValue': 'True',
                     'Name': 'BalanceDataset',
                     'Type': 'String'},
                    {'DefaultValue': 0.9,
                     'Name': 'TrainSplitPercentage',
                     'Type': 'Float'},
                    {'DefaultValue': 0.05,
                     'Name': 'ValidationSplitPercentage',
                     'Type': 'Float'},
                    {'DefaultValue': 0.05,
                     'Name': 'TestSplitPercentage',
                     'Type': 'Float'},
                    {'DefaultValue': 'reviews-feature-store-1662274772',
                     'Name': 'FeatureStoreOfflinePrefix',
                     'Type': 'String'},
                    {'DefaultValue': 'reviews-feature-group-1662274772',
                     'Name': 'FeatureGroupName',
                     'Type': 'String'},
                    {'DefaultValue': 3, 'Name': 'Epochs', 'Type': 'Integer'},
                    {'DefaultValue': 1e-05,
                     'Name': 'LearningRate',
                     'Type': 'Float'},
                    {'DefaultValue': 64,
                     'Name': 'TrainBatchSize',
                     'Type': 'Integer'},
                    {'DefaultValue': 50,
                     'Name': 'TrainStepsPerEpoch',
                     'Type': 'Integer'},
                    {'DefaultValue': 64,
                     'Name': 'ValidationBatchSize',
                     'Type': 'Integer'},
                    {'DefaultValue': 50,
                     'Name': 'ValidationStepsPerEpoch',
                     'Type': 'Integer'},
                    {'DefaultValue': 'False',
                     'Name': 'FreezeBertLayer',
                     'Type': 'String'},
                    {'DefaultValue': 42, 'Name': 'Seed', 'Type': 'Integer'},
                    {'DefaultValue': 1,
                     'Name': 'TrainInstanceCount',
                     'Type': 'Integer'},
                    {'DefaultValue': 'ml.c5.9xlarge',
                     'Name': 'TrainInstanceType',
                     'Type': 'String'},
                    {'DefaultValue': 256,
                     'Name': 'TrainVolumeSize',
                     'Type': 'Integer'},
                    {'DefaultValue': 'File', 'Name': 'InputMode', 'Type': 'St
ring'},
                    {'DefaultValue': 'True',
                     'Name': 'RunValidation',
                     'Type': 'String'},
                    {'DefaultValue': 0.33,
```

```
                 'Name': 'MinAccuracyValue',
                 'Type': 'Float'},
                {'DefaultValue': 'PendingManualApproval',
                 'Name': 'ModelApprovalStatus',
                 'Type': 'String'},
                {'DefaultValue': 'ml.m5.large',
                 'Name': 'DeployInstanceType',
                 'Type': 'String'},
                {'DefaultValue': 1,
                 'Name': 'DeployInstanceCount',
                 'Type': 'Integer'}],
 'Steps': [{'Arguments': {'AppSpecification': {'ContainerArguments': ['--
train-split-percentage',
                                                                      '0.
9',
                                                                      '--
validation-split-percentage',
                                                                      '0.
05',
                                                                      '--
test-split-percentage',
                                                                      '0.
05',
                                                                      '--
balance-dataset',
                                                                      'Tr
ue',
                                                                      '--
max-seq-length',
                                                                      '12
8',
                                                                      '--
feature-store-offline-prefix',
                                                                      're
views-feature-store-1662274772',
                                                                      '--
feature-group-name',
                                                                      're
views-feature-group-1662274772'],
                                                 'ContainerEntrypoint': ['p
ython3',
                                                                       '/
opt/ml/processing/input/code/prepare_data.py'],
                                                 'ImageUri': '683313688378.
dkr.ecr.us-east-1.amazonaws.com/sagemaker-scikit-learn:0.23-1-cpu-py3'},
                          'Environment': {'AWS_DEFAULT_REGION': 'us-east-
1'},
                          'ProcessingInputs': [{'AppManaged': False,
                                                'InputName': 'raw-input-d
ata',
                                                'S3Input': {'LocalPath':
'/opt/ml/processing/input/data/',
                                                            'S3Compressio
nType': 'None',
                                                            'S3DataDistri
butionType': 'ShardedByS3Key',
                                                            'S3DataType':
'S3Prefix',
                                                            'S3InputMode'
: 'File',
                                                            'S3Uri': {'Ge
```

t': 'Parameters.InputData'}}},
{'AppManaged': False,
'InputName': 'code',
'S3Input': {'LocalPath':
'/opt/ml/processing/input/code',
'S3CompressionType': 'None',
'S3DataDistributionType': 'FullyReplicated',
'S3DataType': 'S3Prefix',
'S3InputMode': 'File',
'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker-scikit-learn-2022-09-04-06-59-36-012/input/code/prepare_data.py'}}],
'ProcessingOutputConfig': {'Outputs': [{'AppManaged': False,
'OutputName': 'sentiment-train',
'S3Output': {'LocalPath': '/opt/ml/processing/output/sentiment/train',
'S3UploadMode': 'EndOfJob',
'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker-scikit-learn-2022-09-04-06-59-33-489/output/sentiment-train'}},
{'AppManaged': False,
'OutputName': 'sentiment-validation',
'S3Output': {'LocalPath': '/opt/ml/processing/output/sentiment/validation',
'S3UploadMode': 'EndOfJob',
'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker-scikit-learn-2022-09-04-06-59-33-489/output/sentiment-validation'}},
{'AppManaged': False,
'OutputName': 'sentiment-test',
'S3Output': {'LocalPath': '/opt/ml/processing/output/sentiment/test',
'S3UploadMode': 'EndOfJob',
'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker-scikit-learn-2022-09-04-06-59-33-489/output/sentiment-test'}}]},
'ProcessingResources': {'ClusterConfig': {'InstanceCount': {'Get': 'Parameters.ProcessingInstanceCount'},
'InstanceType': {'Get': 'Parameters.ProcessingInstanceType'},
'VolumeSizeInGB': 30}},
'RoleArn': 'arn:aws:iam::141881372941:role/sagemaker-studio-vpc-firewall-us-east-1-sagemaker-execution-role'},
'Name': 'Processing',
'Type': 'Processing'},
{'Arguments': {'AlgorithmSpecification': {'EnableSageMakerMetricsTimeSeries': True,

```
                                                        'MetricDefinitions':
[{'Name': 'validation:loss',

'Regex': 'val_loss: '

'([0-9.]+)'},

{'Name': 'validation:accuracy',

'Regex': 'val_acc: '

'([0-9.]+)'}],
                                                        'TrainingImage': '76
3104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-training:1.6.0-cpu-py3
',
                                                        'TrainingInputMode':
{'Get': 'Parameters.InputMode'}},
                           'DebugHookConfig': {'CollectionConfigurations':
[],
                                                 'S3OutputPath': 's3://sagem
aker-us-east-1-141881372941/'},
                           'HyperParameters': {'epochs': '3',
                                                 'freeze_bert_layer': '"Fals
e"',
                                                 'learning_rate': '1e-05',
                                                 'max_seq_length': '128',
                                                 'run_validation': '"True"',
                                                 'sagemaker_container_log_le
vel': '20',
                                                 'sagemaker_job_name': '"pyt
orch-training-2022-09-04-06-59-36-136"',
                                                 'sagemaker_program': '"trai
n.py"',
                                                 'sagemaker_region': '"us-ea
st-1"',
                                                 'sagemaker_submit_directory
': '"s3://sagemaker-us-east-1-141881372941/pytorch-training-2022-09-04-06
-59-36-136/source/sourcedir.tar.gz"',
                                                 'seed': '42',
                                                 'train_batch_size': '64',
                                                 'train_steps_per_epoch': '5
0',
                                                 'validation_batch_size': '6
4',
                                                 'validation_steps_per_epoch
': '50'},
                           'InputDataConfig': [{'ChannelName': 'train',
                                                  'ContentType': 'text/csv',
                                                  'DataSource': {'S3DataSour
ce': {'S3DataDistributionType': 'FullyReplicated',

'S3DataType': 'S3Prefix',

'S3Uri': {'Get': "Steps.Processing.ProcessingOutputConfig.Outputs['sentim
ent-train'].S3Output.S3Uri"}}}},
                                                 {'ChannelName': 'validation
',
                                                  'ContentType': 'text/csv',
                                                  'DataSource': {'S3DataSour
ce': {'S3DataDistributionType': 'FullyReplicated',
```

'S3DataType': 'S3Prefix',

'S3Uri': {'Get': "Steps.Processing.ProcessingOutputConfig.Outputs['sentim
ent-validation'].S3Output.S3Uri"}}}}],
'OutputDataConfig': {'S3OutputPath': 's3://sage
maker-us-east-1-141881372941/'},
'ProfilerConfig': {'S3OutputPath': 's3://sagema
ker-us-east-1-141881372941/'},
'ProfilerRuleConfigurations': [{'RuleConfigurat
ionName': 'ProfilerReport-1662274776',
'RuleEvaluatorI
mage': '503895931360.dkr.ecr.us-east-1.amazonaws.com/sagemaker-debugger-r
ules:latest',
'RuleParameters
': {'rule_to_invoke': 'ProfilerReport'}}],
'ResourceConfig': {'InstanceCount': {'Get': 'Pa
rameters.TrainInstanceCount'},
'InstanceType': {'Get': 'Par
ameters.TrainInstanceType'},
'VolumeSizeInGB': {'Get': 'P
arameters.TrainVolumeSize'}},
'RoleArn': 'arn:aws:iam::141881372941:role/sage
maker-studio-vpc-firewall-us-east-1-sagemaker-execution-role',
'StoppingCondition': {'MaxRuntimeInSeconds': 86
400}},
'CacheConfig': {'Enabled': True, 'ExpireAfter': 'PT1H'},
'Name': 'Train',
'Type': 'Training'},
{'Arguments': {'AppSpecification': {'ContainerArguments': ['--
max-seq-length',
'12
8'],
'ContainerEntrypoint': ['p
ython3',
'/
opt/ml/processing/input/code/evaluate_model_metrics.py'],
'ImageUri': '683313688378.
dkr.ecr.us-east-1.amazonaws.com/sagemaker-scikit-learn:0.23-1-cpu-py3'},
'Environment': {'AWS_DEFAULT_REGION': 'us-east-
1'},
'ProcessingInputs': [{'AppManaged': False,
'InputName': 'input-1',
'S3Input': {'LocalPath':
'/opt/ml/processing/input/model',
'S3Compressio
nType': 'None',
'S3DataDistri
butionType': 'FullyReplicated',
'S3DataType':
'S3Prefix',
'S3InputMode'
: 'File',
'S3Uri': {'Ge
t': 'Steps.Train.ModelArtifacts.S3ModelArtifacts'}}},
{'AppManaged': False,
'InputName': 'input-2',
'S3Input': {'LocalPath':
'/opt/ml/processing/input/data',
'S3Compressio
nType': 'None',
'S3DataDistri

butionType': 'FullyReplicated',
'S3DataType': 'S3Prefix',
'S3InputMode': 'File',
'S3Uri': {'Get': "Steps.Processing.ProcessingOutputConfig.Outputs['sentiment-test'].S3Output.S3Uri"}}},
{'AppManaged': False,
'InputName': 'code',
'S3Input': {'LocalPath': '/opt/ml/processing/input/code',
'S3CompressionType': 'None',
'S3DataDistributionType': 'FullyReplicated',
'S3DataType': 'S3Prefix',
'S3InputMode': 'File',
'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker-scikit-learn-2022-09-04-06-59-36-607/input/code/evaluate_model_metrics.py'}}],
'ProcessingOutputConfig': {'Outputs': [{'AppManaged': False,
'OutputName': 'metrics',
'S3Output': {'LocalPath': '/opt/ml/processing/output/metrics/',
'S3UploadMode': 'EndOfJob',
'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker-scikit-learn-2022-09-04-06-59-35-126/output/metrics'}}]},
'ProcessingResources': {'ClusterConfig': {'InstanceCount': {'Get': 'Parameters.ProcessingInstanceCount'},
'InstanceType': {'Get': 'Parameters.ProcessingInstanceType'},
'VolumeSizeInGB': 30}},
'RoleArn': 'arn:aws:iam::141881372941:role/sagemaker-studio-vpc-firewall-us-east-1-sagemaker-execution-role',
'StoppingCondition': {'MaxRuntimeInSeconds': 7200}},
'Name': 'EvaluateModel',
'PropertyFiles': [{'FilePath': 'evaluation.json',
'OutputName': 'metrics',
'PropertyFileName': 'EvaluationReport'}],
'Type': 'Processing'},
{'Arguments': {'Conditions': [{'LeftValue': {'Std:JsonGet': {'Path': 'metrics.accuracy.value',
'PropertyFile': {'Get': 'Steps.EvaluateModel.PropertyFiles.EvaluationReport'}}},
'RightValue': {'Get': 'Parameters.MinAccuracyValue'},
'Type': 'GreaterThanOrEqualTo'}],
'ElseSteps': [],
'IfSteps': [{'Arguments': {'InferenceSpecification': {'Containers': [{'Image': '763104351884.dkr.ecr.us-east-1.amazonaws

```
.com/pytorch-inference:1.6.0-cpu-py36',

'ModelDataUrl': {'Get': 'Steps.Train.ModelArtifacts.S3ModelArtifacts'}}],

'SupportedContentTypes': ['application/jsonlines'],

'SupportedRealtimeInferenceInstanceTypes': [{'Get': 'Parameters.DeployIns
tanceType'}],

'SupportedResponseMIMETypes': ['application/jsonlines'],

'SupportedTransformInstanceTypes': [{'Get': 'Parameters.DeployInstanceTyp
e'}]},
                                                            'ModelApprovalStatus
': {'Get': 'Parameters.ModelApprovalStatus'},
                                                            'ModelMetrics': {'Mo
delQuality': {'Statistics': {'ContentType': 'application/json',

'S3Uri': 's3://sagemaker-us-east-1-141881372941/sagemaker-scikit-learn-20
22-09-04-06-59-35-126/output/metrics/evaluation.json'}}},
                                                            'ModelPackageGroupNa
me': 'BERT-Reviews-1662274772'},
                                          'Name': 'RegisterModel',
                                          'Type': 'RegisterModel'},
                                         {'Arguments': {'ExecutionRoleArn':
'arn:aws:iam::141881372941:role/sagemaker-studio-vpc-firewall-us-east-1-s
agemaker-execution-role',
                                                            'PrimaryContainer':
{'Environment': {},

'Image': '763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:
1.6.0-cpu-py36',

'ModelDataUrl': {'Get': 'Steps.Train.ModelArtifacts.S3ModelArtifacts'}}},
                                          'Name': 'CreateModel',
                                          'Type': 'Model'}]},
             'Name': 'AccuracyCondition',
             'Type': 'Condition'}],
 'Version': '2020-12-01'}
```

Ignore the `WARNING` *below*

Create pipeline using the `create` method and then print the Amazon Resource Name (ARN) of it.

In [39]:
```
response = pipeline.create(role_arn=role)

pipeline_arn = response["PipelineArn"]
print(pipeline_arn)
```

```
No finished training job found associated with this estimator. Please mak
e sure this estimator is only used for building workflow config
arn:aws:sagemaker:us-east-1:141881372941:pipeline/bert-pipeline-166227477
2
```

Ignore the `WARNING` *^^ above ^^*

7.2. Start Pipeline

Let's submit our pipeline definition to the Amazon SageMaker Pipeline service. The role passed in will be used by the service to create all the jobs defined in the steps. You will start the pipeline using the parameters passed into the `start()` function.

In [40]:
```
execution = pipeline.start(
    parameters=dict(
        InputData=raw_input_data_s3_uri,
        ProcessingInstanceCount=1,
        ProcessingInstanceType='ml.c5.2xlarge',
        MaxSeqLength=128,
        BalanceDataset='True',
        TrainSplitPercentage=0.9,
        ValidationSplitPercentage=0.05,
        TestSplitPercentage=0.05,
        FeatureStoreOfflinePrefix='reviews-feature-store-'+str(timestamp)
        FeatureGroupName='reviews-feature-group-'+str(timestamp),
        Epochs=3,
        LearningRate=0.000012,
        TrainBatchSize=64,
        TrainStepsPerEpoch=50,
        ValidationBatchSize=64,
        ValidationStepsPerEpoch=64,
        FreezeBertLayer='False',
        Seed=42,
        TrainInstanceCount=1,
        TrainInstanceType='ml.c5.9xlarge',
        TrainVolumeSize=256,
        InputMode='File',
        RunValidation='True',
        MinAccuracyValue=0.01,
        ModelApprovalStatus='PendingManualApproval',
        DeployInstanceType='ml.m5.large',
        DeployInstanceCount=1
    )
)

print(execution.arn)
```

```
arn:aws:sagemaker:us-east-1:141881372941:pipeline/bert-pipeline-166227477
2/execution/dh54d7we8u88
```

7.3. Wait for pipeline execution

Now you can describe execution instance and list the steps in the execution to find out more about the execution.

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

execution_run = execution.describe()
pprint(execution_run)
```

```
{'CreatedBy': {'DomainId': 'd-4ftrlzf4sfyd',
               'UserProfileArn': 'arn:aws:sagemaker:us-east-1:14188137294
1:user-profile/d-4ftrlzf4sfyd/sagemaker-user-profile-us-east-1',
               'UserProfileName': 'sagemaker-user-profile-us-east-1'},
 'CreationTime': datetime.datetime(2022, 9, 4, 6, 59, 37, 893000, tzinfo=
tzlocal()),
 'LastModifiedBy': {'DomainId': 'd-4ftrlzf4sfyd',
                    'UserProfileArn': 'arn:aws:sagemaker:us-east-1:141881
372941:user-profile/d-4ftrlzf4sfyd/sagemaker-user-profile-us-east-1',
                    'UserProfileName': 'sagemaker-user-profile-us-east-1'
},
 'LastModifiedTime': datetime.datetime(2022, 9, 4, 6, 59, 37, 893000, tzi
nfo=tzlocal()),
 'PipelineArn': 'arn:aws:sagemaker:us-east-1:141881372941:pipeline/bert-p
ipeline-1662274772',
 'PipelineExecutionArn': 'arn:aws:sagemaker:us-east-1:141881372941:pipeli
ne/bert-pipeline-1662274772/execution/dh54d7we8u88',
 'PipelineExecutionDisplayName': 'execution-1662274777995',
 'PipelineExecutionStatus': 'Executing',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '815',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Sun, 04 Sep 2022 06:59:37
GMT',
                                      'x-amzn-requestid': 'dea8a727-82a4-
4e49-af16-90d140f2d1fa'},
                       'HTTPStatusCode': 200,
                       'RequestId': 'dea8a727-82a4-4e49-af16-90d140f2d1fa'
,
                       'RetryAttempts': 0}}
```

Print the execution display name and its ARN:

In [42]:
```
execution_run_name = execution_run['PipelineExecutionDisplayName']
print(execution_run_name)
```

```
execution-1662274777995
```

In [43]:
```
pipeline_execution_arn = execution_run['PipelineExecutionArn']
print(pipeline_execution_arn)
```

```
arn:aws:sagemaker:us-east-1:141881372941:pipeline/bert-pipeline-166227477
2/execution/dh54d7we8u88
```

7.4. Describe completed pipeline

Wait for the first step to start running and print the information about it:

In [44]:
```
import time

time.sleep(30)

execution.list_steps()
```

Out[44]:
```
[{'StepName': 'Processing',
  'StartTime': datetime.datetime(2022, 9, 4, 6, 59, 38, 623000, tzinfo=tz
local()),
  'StepStatus': 'Executing',
  'AttemptCount': 0,
  'Metadata': {'ProcessingJob': {'Arn': 'arn:aws:sagemaker:us-east-1:1418
81372941:processing-job/pipelines-dh54d7we8u88-processing-o6skn85hoa'}}}]
```

7.5. Wait for the pipeline to complete

To get the information about the pipeline execution you can use low-level service client of the boto3 session. It is also useful for other operations that you will see below.

In the code below you will be observing the pipeline execution summary and waiting for the execution status to change from `Executing` to `Succeeded`.

This cell will take approximately 30-45 minutes to run.

In [45]:
```
%%time

import time
from pprint import pprint

sm = boto3.Session().client(service_name='sagemaker', region_name=region)

executions_response = sm.list_pipeline_executions(PipelineName=pipeline_n
pipeline_execution_status = executions_response[0]['PipelineExecutionStat
print(pipeline_execution_status)

while pipeline_execution_status=='Executing':
    try:
        executions_response = sm.list_pipeline_executions(PipelineName=pi
        pipeline_execution_status = executions_response[0]['PipelineExecu
    except Exception as e:
        print('Please wait...')
        time.sleep(30)

pprint(executions_response)
```

```
Executing
Please wait...
Please wait...
Please wait...
```

```
Succeeded
```

In [47]:
```
pipeline_execution_arn = executions_response[0]['PipelineExecutionArn']
print(pipeline_execution_arn)
```

```
arn:aws:sagemaker:us-east-1:141881372941:pipeline/bert-pipeline-166227477
2/execution/dh54d7we8u88
```

8. Evaluate the model

8.1. Describe evaluation metrics

Examine the resulting model evaluation after the pipeline completes. Download the resulting evaluation.json file from S3 and print the report.

In [48]:
```
processing_job_name = None

# pull the processing step name
for execution_step in reversed(execution.list_steps()):
    if execution_step['StepName'] == 'Processing':
        processing_job_name=execution_step['Metadata']['ProcessingJob']['

# get the description of the processing job
describe_transform_processing_job_response = sm.describe_processing_job(P

# get the output S3 path
transform_output_s3_uri = describe_transform_processing_job_response['Pro
print('Transform output {}'.format(transform_output_s3_uri))
```

```
Transform output s3://sagemaker-us-east-1-141881372941/sagemaker-scikit-l
earn-2022-09-04-06-59-33-489/output/sentiment-train
```

In [49]:
```
# list the files in the resulting output S3 path
!aws s3 ls --recursive $transform_output_s3_uri
```

```
2022-09-04 07:13:35    4876932 sagemaker-scikit-learn-2022-09-04-06-59-33
-489/output/sentiment-train/part-algo-1-womens_clothing_ecommerce_reviews
.tsv
```

Exercise 7

Pull the name of the model-evaluation step and then get the S3 path of the evaluation metrics, which will contain the evaluation report.

Instructions: Find the execution step with the step name `EvaluateModel` following the example above.

In [50]:
```
processing_job_name = None

for execution_step in reversed(execution.list_steps()):
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    if execution_step['StepName'] == 'EvaluateModel': # Replace all None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
        processing_job_name=execution_step['Metadata']['ProcessingJob']['

describe_evaluation_processing_job_response = sm.describe_processing_job(

evaluation_metrics_s3_uri = describe_evaluation_processing_job_response['
print('Evaluation output {}'.format(evaluation_metrics_s3_uri))
```

```
Evaluation output s3://sagemaker-us-east-1-141881372941/sagemaker-scikit-
learn-2022-09-04-06-59-35-126/output/metrics
```

8.2. Review the evaluation report

Download the evaluation report and print the accuracy.

In [51]:
```
from pprint import pprint

evaluation_json = sagemaker.s3.S3Downloader.read_file("{}/evaluation.json
    evaluation_metrics_s3_uri
))

pprint(json.loads(evaluation_json))
```

```
{'metrics': {'accuracy': {'value': 0.7475728155339806}}}
```

8.3. List pipeline artifacts

Exercise 8

Find and print the ARN and job name of the training job.

Instructions: Find the execution step with the step name `Train` following the example above.

In [52]:
```
training_job_arn=None

for execution_step in execution.list_steps():
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    if execution_step['StepName'] == 'Train': # Replace all None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
        training_job_arn = execution_step['Metadata']['TrainingJob']['Arn
        pprint(execution_step)
        break
print('Training job ARN: {}'.format(training_job_arn))

training_job_name = training_job_arn.split('/')[-1]
print('Training job Name: {}'.format(training_job_name))
```

```
{'AttemptCount': 0,
 'EndTime': datetime.datetime(2022, 9, 4, 7, 34, 44, 166000, tzinfo=tzloc
al()),
 'Metadata': {'TrainingJob': {'Arn': 'arn:aws:sagemaker:us-east-1:1418813
72941:training-job/pipelines-dh54d7we8u88-train-aawqvjpxen'}},
 'StartTime': datetime.datetime(2022, 9, 4, 7, 13, 41, 911000, tzinfo=tzl
ocal()),
 'StepName': 'Train',
 'StepStatus': 'Succeeded'}
Training job ARN: arn:aws:sagemaker:us-east-1:141881372941:training-job/p
ipelines-dh54d7we8u88-train-aawqvjpxen
Training job Name: pipelines-dh54d7we8u88-train-aawqvjpxen
```

Using similar approach you can find and print the pipeline artifacts.

In [53]:
```
processing_job_name=None
training_job_name=None
```

In [54]:
```
import time
from sagemaker.lineage.visualizer import LineageTableVisualizer

viz = LineageTableVisualizer(sagemaker.session.Session())

for execution_step in reversed(execution.list_steps()):
    pprint(execution_step)
    if execution_step['StepName'] == 'Processing':
        processing_job_name=execution_step['Metadata']['ProcessingJob']['
        print('Processing job name: {}'.format(processing_job_name))
        display(viz.show(processing_job_name=processing_job_name))
    elif execution_step['StepName'] == 'Train':
        training_job_name=execution_step['Metadata']['TrainingJob']['Arn'
        print('Training job name: {}'.format(training_job_name))
        display(viz.show(training_job_name=training_job_name))
    else:
        display(viz.show(pipeline_execution_step=execution_step))
        time.sleep(5)
```

```
{'AttemptCount': 0,
 'EndTime': datetime.datetime(2022, 9, 4, 7, 13, 41, 387000, tzinfo=tzloc
al()),
 'Metadata': {'ProcessingJob': {'Arn': 'arn:aws:sagemaker:us-east-1:14188
1372941:processing-job/pipelines-dh54d7we8u88-processing-o6skn85hoa'}},
 'StartTime': datetime.datetime(2022, 9, 4, 6, 59, 38, 623000, tzinfo=tzl
ocal()),
 'StepName': 'Processing',
 'StepStatus': 'Succeeded'}
Processing job name: pipelines-dh54d7we8u88-processing-o6skn85hoa
```

	Name/Source	Direction	Type	Association Type	Lineage Type
0	s3://...-06-59-36-833/input/code/prepare_data.py	Input	DataSet	ContributedTo	artifact
1	s3://dlai-practical-data-science/data/raw/	Input	DataSet	ContributedTo	artifact
2	68331...om/sagemaker-scikit-learn:0.23-1-cpu-py3	Input	Image	ContributedTo	artifact
3	s3://...09-04-06-59-33-489/output/sentiment-test	Output	DataSet	Produced	artifact
4	s3://...06-59-33-489/output/sentiment-validation	Output	DataSet	Produced	artifact
5	s3://...9-04-06-59-33-489/output/sentiment-train	Output	DataSet	Produced	artifact

```
{'AttemptCount': 0,
 'EndTime': datetime.datetime(2022, 9, 4, 7, 34, 44, 166000, tzinfo=tzloc
al()),
 'Metadata': {'TrainingJob': {'Arn': 'arn:aws:sagemaker:us-east-1:1418813
72941:training-job/pipelines-dh54d7we8u88-train-aawqvjpxen'}},
 'StartTime': datetime.datetime(2022, 9, 4, 7, 13, 41, 911000, tzinfo=tzl
ocal()),
 'StepName': 'Train',
 'StepStatus': 'Succeeded'}
Training job name: pipelines-dh54d7we8u88-train-aawqvjpxen
```

	Name/Source	Direction	Type	Association Type	Lineage Type
0	s3://...06-59-33-489/output/sentiment-validation	Input	DataSet	ContributedTo	artifact
1	s3://...9-04-06-59-33-489/output/sentiment-train	Input	DataSet	ContributedTo	artifact
2	76310...onaws.com/pytorch-training:1.6.0-cpu-py3	Input	Image	ContributedTo	artifact
3	s3://...u88-Train-AAWQVJPxeN/output/model.tar.gz	Output	Model	Produced	artifact

```
{'AttemptCount': 0,
 'EndTime': datetime.datetime(2022, 9, 4, 7, 43, 13, 700000, tzinfo=tzloc
al()),
 'Metadata': {'ProcessingJob': {'Arn': 'arn:aws:sagemaker:us-east-1:14188
1372941:processing-job/pipelines-dh54d7we8u88-evaluatemodel-qkca7pf6qk'}}
,
 'StartTime': datetime.datetime(2022, 9, 4, 7, 34, 45, 183000, tzinfo=tzl
ocal()),
 'StepName': 'EvaluateModel',
 'StepStatus': 'Succeeded'}
```

	Name/Source	Direction	Type	Association Type	Lineag Typ
0	s3://...065/input/code/evaluate_model_metrics.py	Input	DataSet	ContributedTo	artifac
1	s3://...09-04-06-59-33-489/output/sentiment-test	Input	DataSet	ContributedTo	artifac
2	s3://...u88-Train-AAWQVJPxeN/output/model.tar.gz	Input	Model	ContributedTo	artifac
3	68331...om/sagemaker-scikit-learn:0.23-1-cpu-py3	Input	Image	ContributedTo	artifac
4	s3://...n-2022-09-04-06-59-35-126/output/metrics	Output	DataSet	Produced	artifac

```
{'AttemptCount': 0,
 'EndTime': datetime.datetime(2022, 9, 4, 7, 43, 14, 577000, tzinfo=tzloc
al()),
 'Metadata': {'Condition': {'Outcome': 'True'}},
 'StartTime': datetime.datetime(2022, 9, 4, 7, 43, 14, 233000, tzinfo=tzl
ocal()),
 'StepName': 'AccuracyCondition',
 'StepStatus': 'Succeeded'}
None
{'AttemptCount': 0,
 'EndTime': datetime.datetime(2022, 9, 4, 7, 43, 16, 157000, tzinfo=tzloc
al()),
 'Metadata': {'Model': {'Arn': 'arn:aws:sagemaker:us-east-1:141881372941:
model/pipelines-dh54d7we8u88-createmodel-681vuhw2fy'}},
 'StartTime': datetime.datetime(2022, 9, 4, 7, 43, 14, 949000, tzinfo=tzl
ocal()),
 'StepName': 'CreateModel',
 'StepStatus': 'Succeeded'}
None
{'AttemptCount': 0,
 'EndTime': datetime.datetime(2022, 9, 4, 7, 43, 16, 2000, tzinfo=tzlocal
()),
 'Metadata': {'RegisterModel': {'Arn': 'arn:aws:sagemaker:us-east-1:14188
1372941:model-package/bert-reviews-1662274772/1'}},
 'StartTime': datetime.datetime(2022, 9, 4, 7, 43, 14, 949000, tzinfo=tzl
ocal()),
 'StepName': 'RegisterModel',
 'StepStatus': 'Succeeded'}
```

	Name/Source	Direction	Type	Association Type	Lineage Type
0	s3://...u88-Train-AAWQVJPxeN/output/model.tar.gz	Input	Model	ContributedTo	artifact
1	76310...aws.com/pytorch-inference:1.6.0-cpu-py36	Input	Image	ContributedTo	artifact
2	bert-reviews-1662274772-1-PendingManualApprova...	Input	Approval	ContributedTo	action
3	BERT-Reviews-1662274772-1662277395-aws-model-p...	Output	ModelGroup	AssociatedWith	context

9. Deploy and test the model

9.1. Approve trained model

The pipeline created a model package version within the specified model package group and an approval status of `PendingManualApproval`. This requires a separate step to manually approve the model before deploying to production.

You can approve the model using the SageMaker Studio UI or programmatically as shown below.

Get the model package ARN.

In [55]:
```
for execution_step in execution.list_steps():
    if execution_step['StepName'] == 'RegisterModel':
        model_package_arn = execution_step['Metadata']['RegisterModel']['
        break
print(model_package_arn)
```

```
arn:aws:sagemaker:us-east-1:141881372941:model-package/bert-reviews-16622
74772/1
```

Update the model package with the `Approved` status to prepare for deployment.

The model must be `Approved` before it can be deployed.

In [56]:
```
model_package_update_response = sm.update_model_package(
    ModelPackageArn=model_package_arn,
    ModelApprovalStatus="Approved",
)

pprint(model_package_update_response)
```

```
{'ModelPackageArn': 'arn:aws:sagemaker:us-east-1:141881372941:model-packa
ge/bert-reviews-1662274772/1',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '102',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Sun, 04 Sep 2022 07:47:51
GMT',
                                      'x-amzn-requestid': 'e55ca522-cf2d-
4b91-8b37-0cda586d1231'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'e55ca522-cf2d-4b91-8b37-0cda586d1231'
,
                      'RetryAttempts': 0}}
```

9.2. Deploy model

Get the model ARN and the model name from it.

In [57]:
```
for execution_step in execution.list_steps():
    print(execution_step['StepName'])
    if execution_step['StepName'] == 'CreateModel':
        model_arn = execution_step['Metadata']['Model']['Arn']
        break
print(model_arn)

model_name = model_arn.split('/')[-1]
print(model_name)
```

```
RegisterModel
CreateModel
arn:aws:sagemaker:us-east-1:141881372941:model/pipelines-dh54d7we8u88-cre
atemodel-681vuhw2fy
pipelines-dh54d7we8u88-createmodel-681vuhw2fy
```

9.3. Create endpoint from registry

Configure the endpoint.

In [58]:
```
endpoint_config_name = 'bert-model-epc-{}'.format(timestamp)
print(endpoint_config_name)

create_endpoint_config_response = sm.create_endpoint_config(
    EndpointConfigName = endpoint_config_name,
    ProductionVariants=[{
        'InstanceType':'ml.m5.xlarge',
        'InitialVariantWeight':1,
        'InitialInstanceCount':1,
        'ModelName': model_name,
        'VariantName':'AllTraffic'}])
```

```
bert-model-epc-1662274772
```

Create the endpoint.

In [59]:
```
pipeline_endpoint_name = 'bert-model-ep-{}'.format(timestamp)
print("EndpointName={}".format(pipeline_endpoint_name))

create_endpoint_response = sm.create_endpoint(
    EndpointName=pipeline_endpoint_name,
    EndpointConfigName=endpoint_config_name)
print(create_endpoint_response['EndpointArn'])
```

```
EndpointName=bert-model-ep-1662274772
arn:aws:sagemaker:us-east-1:141881372941:endpoint/bert-model-ep-166227477
2
```

In [60]:
```
from IPython.core.display import display, HTML

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

Review SageMaker REST Endpoint

Wait until the endpoint is deployed.

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

In [61]:
```
%%time

while True:
    try:
        waiter = sm.get_waiter('endpoint_in_service')
        print('Waiting for endpoint to be in `InService`...')
        waiter.wait(EndpointName=pipeline_endpoint_name)
        break;
    except:
        print('Waiting for endpoint...')
        endpoint_status = sm.describe_endpoint(EndpointName=pipeline_endp
        print('Endpoint status: {}'.format(endpoint_status))
        if endpoint_status == 'Failed':
            break
        time.sleep(30)

print('Endpoint deployed.')
```

```
Waiting for endpoint to be in `InService`...
Endpoint deployed.
CPU times: user 119 ms, sys: 10 ms, total: 129 ms
Wall time: 4min 1s
```

Wait until the endpoint ^^ above ^^ is deployed.

9.4. Test model

Predict the `sentiment` with `review_body` samples and review the result:

In [62]:
```
from sagemaker.predictor import Predictor
from sagemaker.serializers import JSONLinesSerializer
from sagemaker.deserializers import JSONLinesDeserializer

inputs = [
    {"features": ["I love this product!"]},
    {"features": ["OK, but not great."]},
    {"features": ["This is not the right product."]},
]

predictor = Predictor(
    endpoint_name=pipeline_endpoint_name,
    serializer=JSONLinesSerializer(),
    deserializer=JSONLinesDeserializer(),
    sagemaker_session=sess
)

predicted_classes = predictor.predict(inputs)

for predicted_class in predicted_classes:
    print("Predicted class {} with probability {}".format(predicted_class
```

```
Predicted class 1 with probability 0.9341263771057129
Predicted class 0 with probability 0.5311968922615051
Predicted class -1 with probability 0.7742885947227478
```

9.5. SageMaker Studio extensions

SageMaker Studio provides a rich set of features to visually inspect SageMaker resources including pipelines, training jobs, and endpoints. Please take time to explore it opening the facet shown in the following image.

Congratulations! You have just deployed an end-to-end pipeline with BERT and SageMaker Pipelines.

Upload the notebook into S3 bucket for grading purposes.

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

In [63]:
```
!aws s3 cp ./C2_W3_Assignment.ipynb s3://$bucket/C2_W3_Assignment_Learner
```

```
upload: ./C2_W3_Assignment.ipynb to s3://sagemaker-us-east-1-141881372941
/C2_W3_Assignment_Learner.ipynb
```

Please go to the main lab window and click on `Submit` button (see the `Finish the lab` section of the instructions).

In [ ]:
```
```

DeepLearning.AI

aws

Advanced Model Training

Machine Learning Workflow

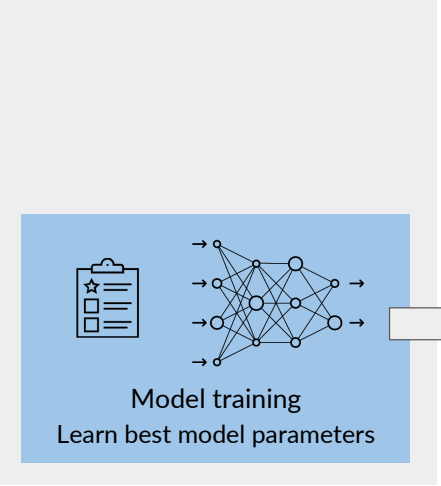
Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Model Tuning

Model Tuning

Model parameters

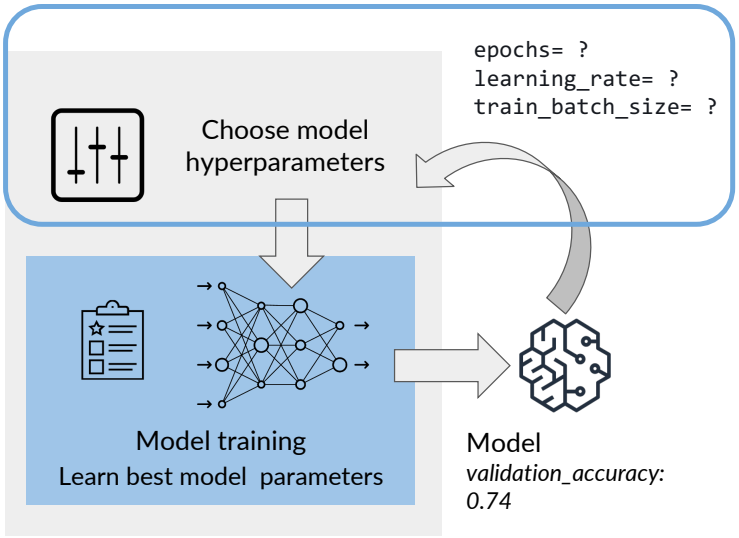
roberta-base
125M parameters



Model Tuning

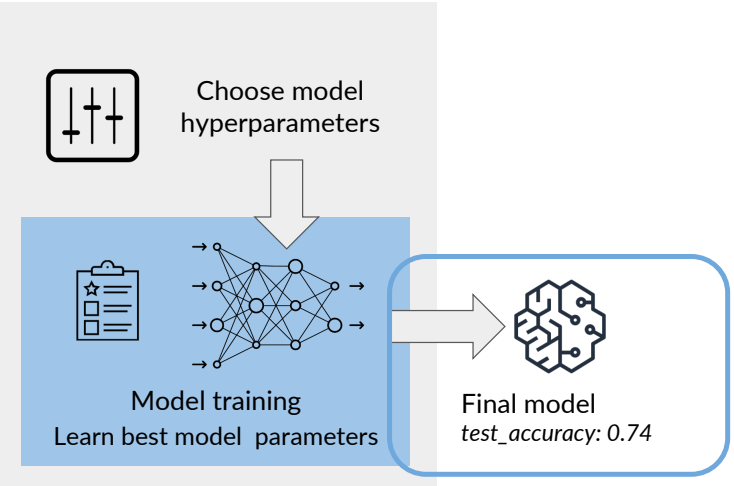
Model hyperparameters

Model parameters



Model Evaluation

"If you can't measure it, you can't improve it."
-- *Peter Drucker*



Manual vs. Automatic Model Tuning

Manual tuning

Automatic model tuning

Popular Algorithms for Automatic Model Tuning

Automatic model tuning

- Grid search
- Random search
- Bayesian optimization
- Hyperband

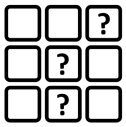
Grid Search

- Define sets of hyperparameters
- Test **every** combination
- Select the best performing hyperparameters

+ Explores all combinations
+ Works for small number of parameters
- Time-consuming
- Doesn't scale to large numbers of parameters

Random Search

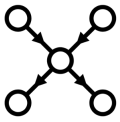
- Define sets of hyperparameters
- Define search space & stop criteria
- Test **random** combinations within search space
- Select the best performing hyperparameters



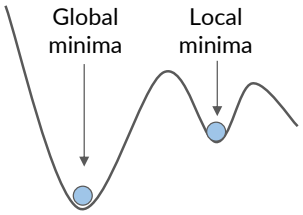
+ Faster compared to grid search
- Might miss better performing hyperparameters

Bayesian Optimization

- Treat HPT like a **regression** problem (surrogate model)
- Start from random hyperparameters
- Narrow down search space around better performing hyperparameters

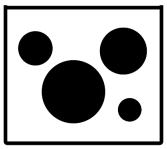


+ More efficient in finding best hyperparameters
- Requires sequential execution
- Might get stuck in local minima



Hyperband

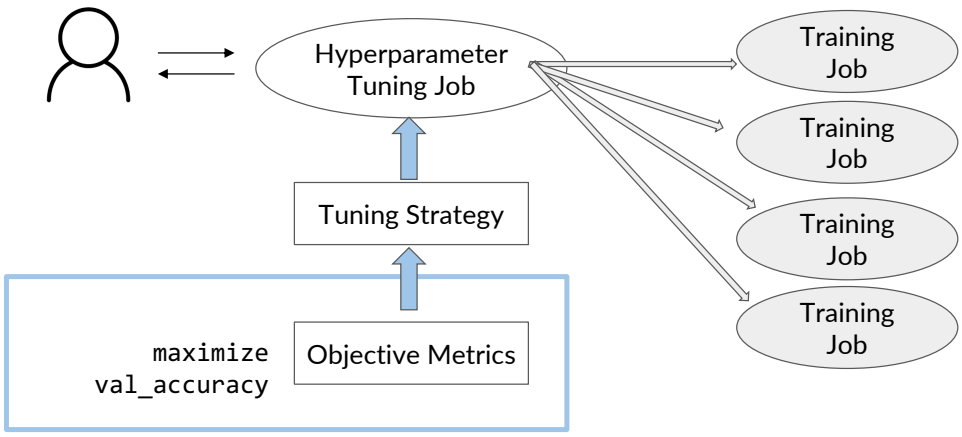
- **Bandit-based** approach
- Start from random hyperparameters
- Explore sets of hyperparameters for few iterations
- Choose best and explore longer
- Repeat until `max_iterations` reached or one candidate left



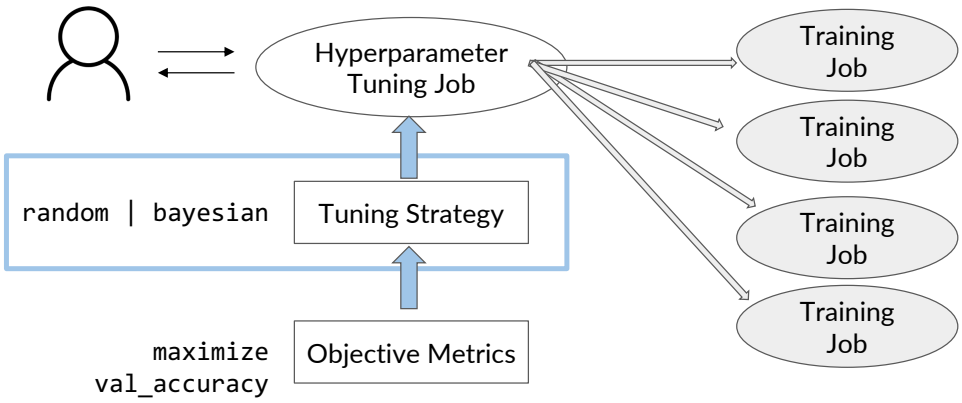
+ Spends time efficiently (explore-exploit theory)
- Might discard good candidates early that converge slowly

Tune a BERT-based Text Classifier

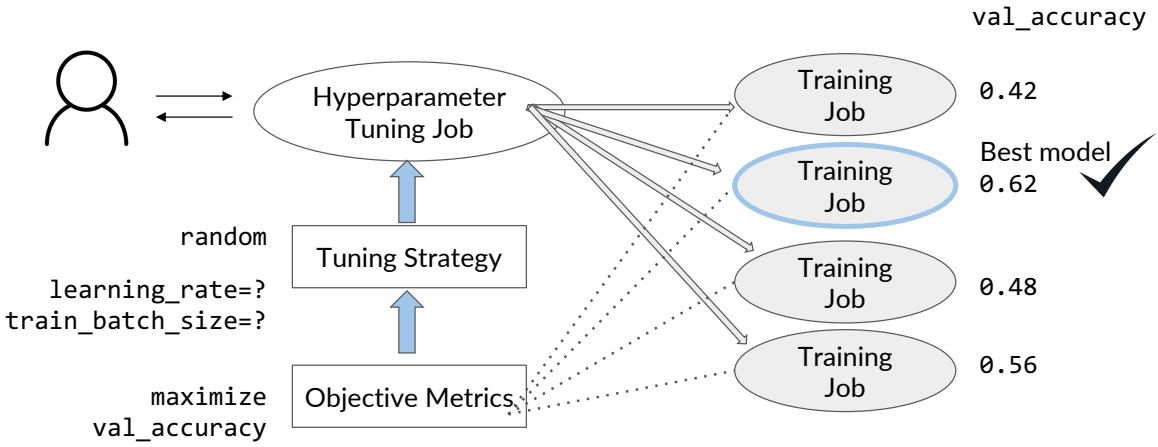
Amazon SageMaker Hyperparameter Tuning (HPT)



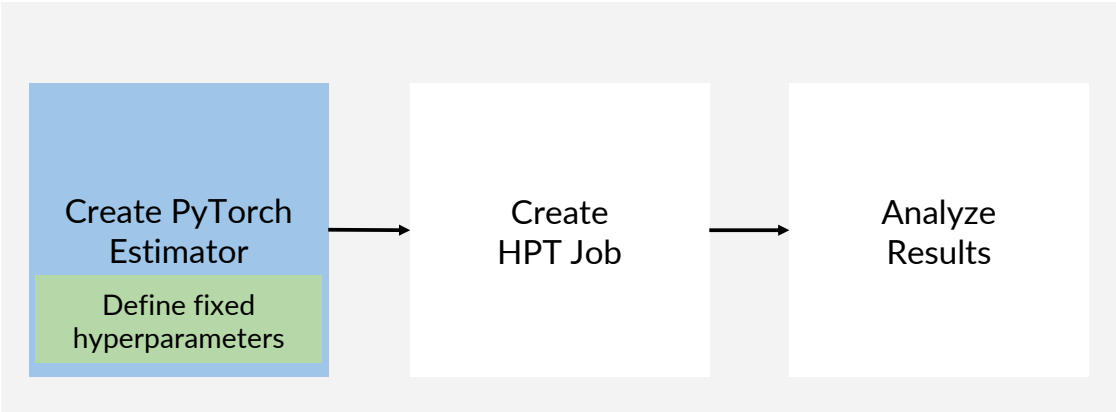
Amazon SageMaker Hyperparameter Tuning (HPT)



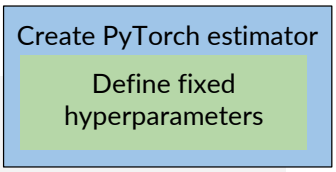
Tune BERT Text Classifier



Steps



Define Fixed Hyperparameters




```
hyperparameters={
     'epochs': 3,
     'train_steps_per_epoch': 50,
     'validation_batch_size': 64,
     'validation_steps_per_epoch': 50,
     'freeze_bert_layer': False,
     'seed': 42,
     'max_seq_length': 64,
     'backend': 'gloo',
     'run_validation': True,
     'run_sample_predictions': False
}
```

Create PyTorch Estimator

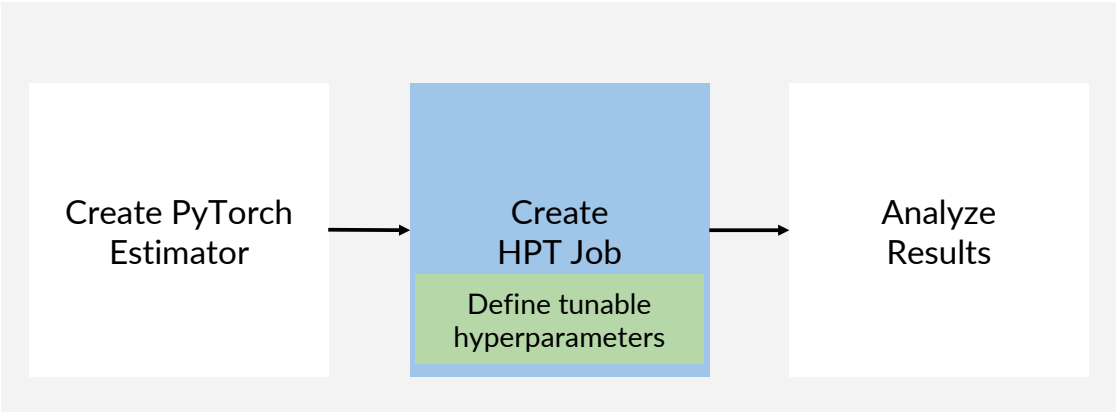
Create PyTorch estimator

Define fixed hyperparameters

```
from sagemaker.pytorch import PyTorch as PyTorchEstimator

estimator = PyTorchEstimator(
    entry_point='train.py',
    ...
    hyperparameters=hyperparameters,

)
```

Fixed hyperparameters in estimator

Steps



Define Tunable Hyperparameters

Create HPT job

Define tunable hyperparameters

```
from sagemaker.tuner import CategoricalParameter
from sagemaker.tuner import ContinuousParameter
from sagemaker.tuner import IntegerParameter

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

Specify parameters

Specify hyperparameter types

Specify ranges

How to Choose Hyperparameter Types

Categorical

```
'train_batch_size':
CategoricalParameter([128, 256])
```

```
'freeze_bert_layer':
CategoricalParameter([True, False])
```

Integer

```
'train_batch_size':
IntegerParameter(16, 1024, scaling_type='Logarithmic')
```

If you need to explore large ranges quickly

How to Choose Hyperparameter Types

Categorical

```
'train_batch_size':
CategoricalParameter([128, 256])
```

```
'freeze_bert_layer':
CategoricalParameter([True, False])
```

Integer

```
'train_batch_size':
IntegerParameter(16, 1024, scaling_type='Logarithmic')
```

Continuous

```
'learning_rate':
ContinuousParameter(0.00001, 0.00005, scaling_type='Linear')
```


Create Amazon SageMaker HPT job

Create HPT job

Define tunable hyperparameters

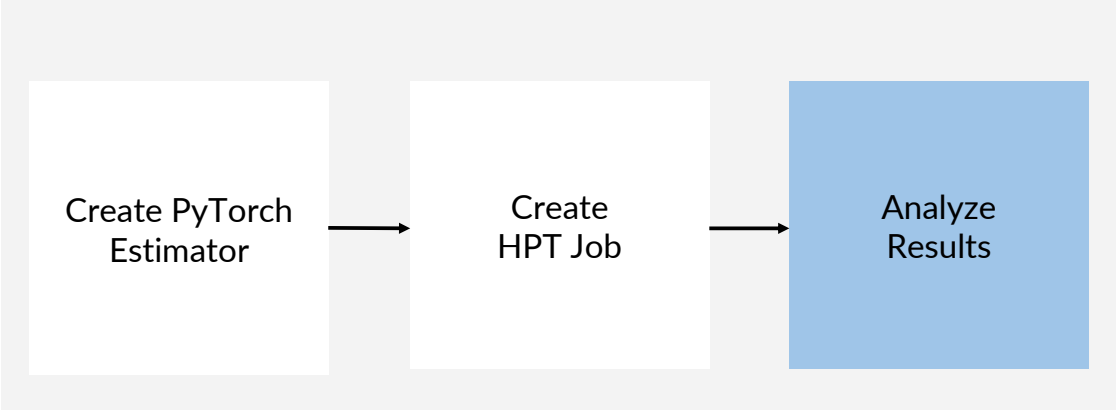
```
from sagemaker.tuner import HyperparameterTuner

tuner = HyperparameterTuner(
    estimator=...,
    hyperparameter_ranges=...,
    objective_type=...,
    objective_metric_name=...,
    strategy=...,
)

tuner.fit(inputs={...}, ...)
```

Pass in estimator

Configure Hyperparameter ranges

Run HPT job with .fit()

Steps



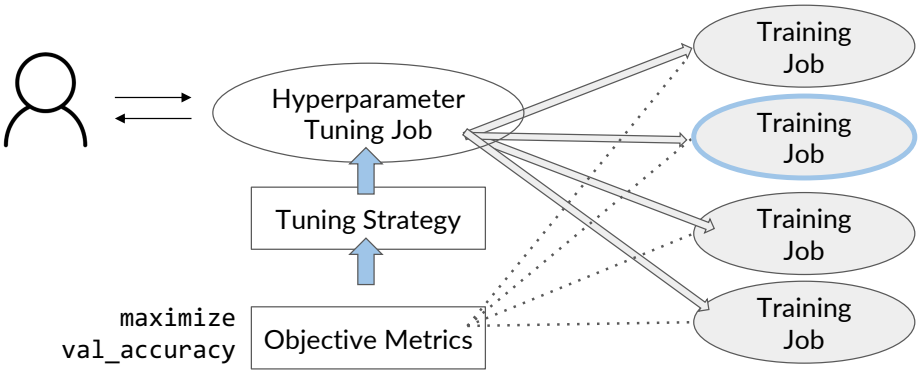
Analyze Results

Analyze Results

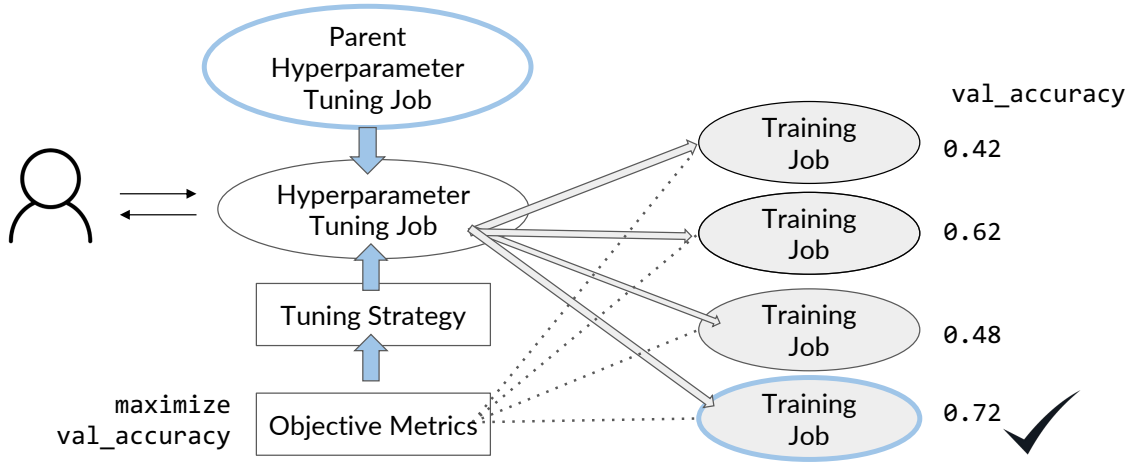
```
df_results = tuner.analytics().dataframe()
```

learning_rate	train_batch_size	TrainingJobName	TrainingJobStatus	FinalObjectiveValue
0.000021	"128"	pytorch-training-210225-1535-001-71394bc3	Completed	44.939999
0.000035	"128"	pytorch-training-210225-1535-002-cf437bad	Completed	41.580002

Warm Start HPT Job



Warm Start HPT Job



Warm Start HPT Job

- IDENTICAL_DATA_AND_ALGORITHM
  - Same input data and training data
  - Update hyperparameter tuning ranges and maximum number of training jobs
- TRANSFER_LEARNING
  - Updated training data and different version of training algorithm

Configure Warm Start

```
from sagemaker.tuner import WarmStartConfig
from sagemaker.tuner import WarmStartTypes

warm_start_config = WarmStartConfig(
     warm_start_type=WarmStartTypes.IDENTICAL_DATA_AND_ALGORITHM,
     parents=<PARENT_TUNING_JOB_NAME>)

tuner = HyperparameterTuner(
    ...
    warm_start_config=warm_start_config)

tuner.fit(...)
```

IDENTICAL_DATA_AND_ALGORITHM or TRANSFER_LEARNING

Specify parent tuning job

Pass warm start config in HyperparameterTuner

Best Practices - SageMaker HyperParameter Tuning

- Select a small number of hyperparameters
- Select a small range for hyperparameters
- Enable warm start
- Enable early stop to save tuning time and costs
- Select a small number of concurrent training jobs

Best Practices - Monitoring Training Resources

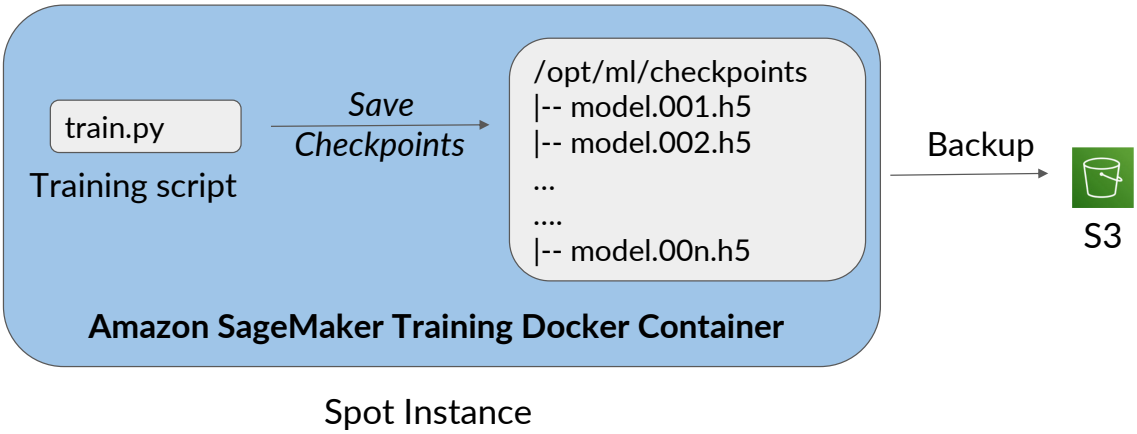
- Right size compute resources
- Requires empirical testing
- Amazon CloudWatch Metrics
- Insights from Amazon SageMaker Debugger

Checkpointing

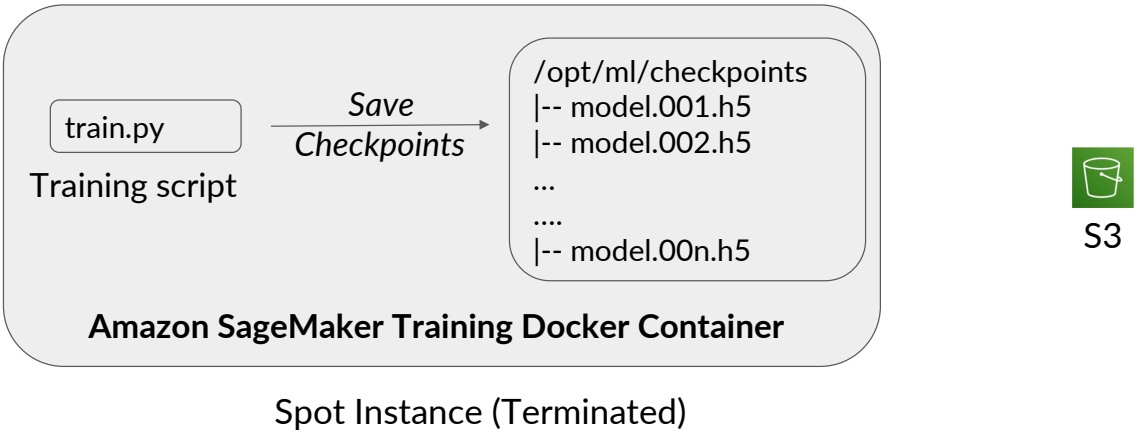
Machine Learning Checkpointing

- Save state of ML models during training
- Checkpoints : Snapshots of the model
  - Model architecture
  - Model weights
  - Training configurations
  - Optimizer
- Frequency and number of checkpoints

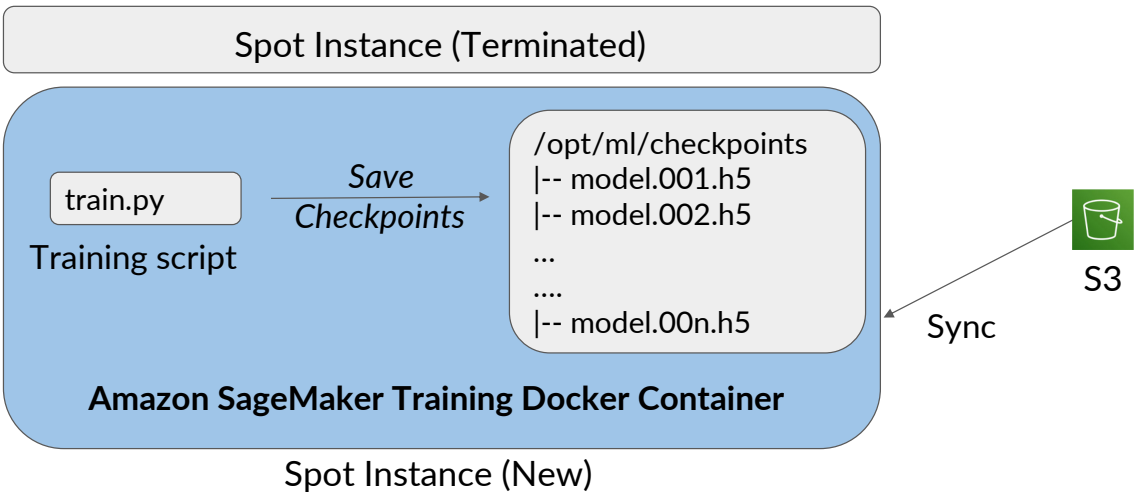
Amazon SageMaker Managed Spot



Amazon SageMaker Managed Spot



Amazon SageMaker Spot Training



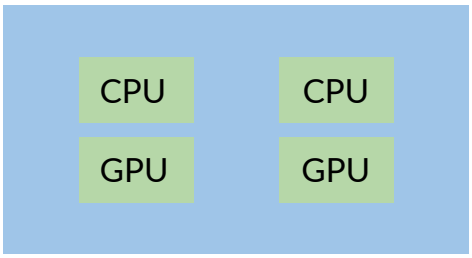
Distributed Training Strategies

Challenges

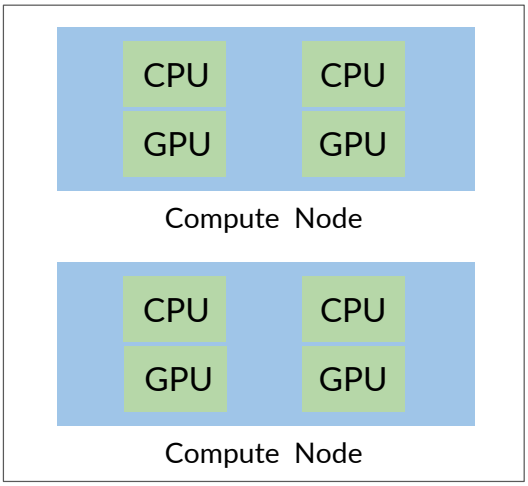
Increased training data volume

Increased model size and complexity

Distributed Training

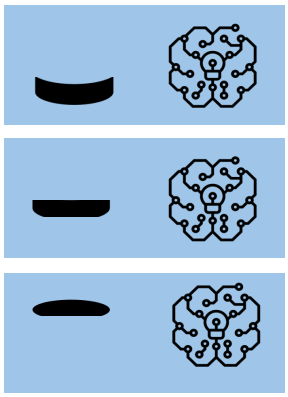


Compute Node

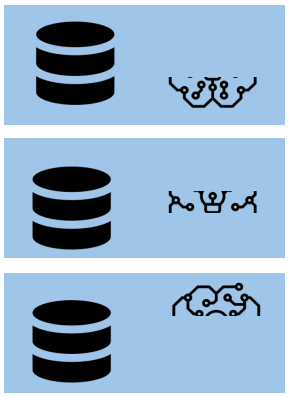


Compute Cluster

Distributed Training Strategies



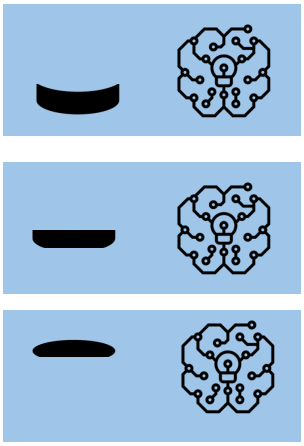
Data Parallelism



Model Parallelism

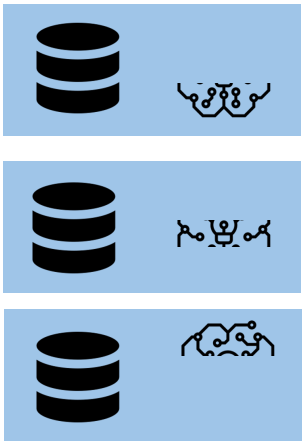
Distributed Training Strategies - Data Parallelism

- Training data split up
- Model replicated on all nodes



Distributed Training Strategies - Model Parallelism

- Training data replicated
- Model split up on all nodes



Amazon SageMaker Estimator

```
from sagemaker.pytorch import PyTorch
estimator = PyTorch(
    entry_point='train.py',
    role=sagemaker.get_execution_role(),
    framework_version='1.6.0',
    py_version='py3',
    instance_count=3,
    instance_type='ml.p3.16xlarge',
    distribution={'smdistributed':{'dataparallel':{enabled': True}}}
)
estimator.fit()
```

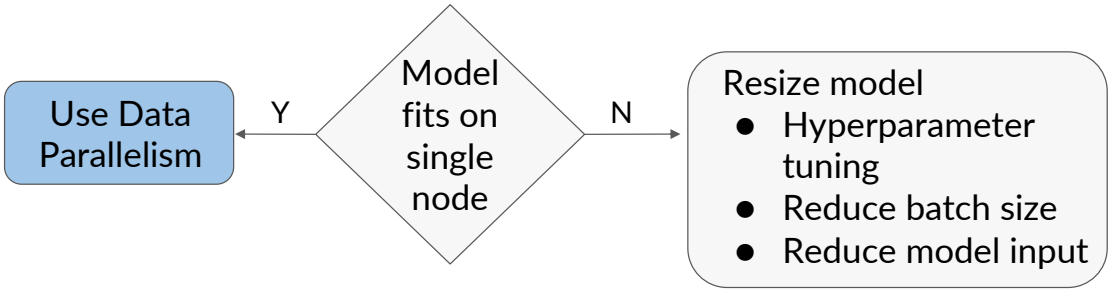
Distribution Strategy

Amazon SageMaker Estimator

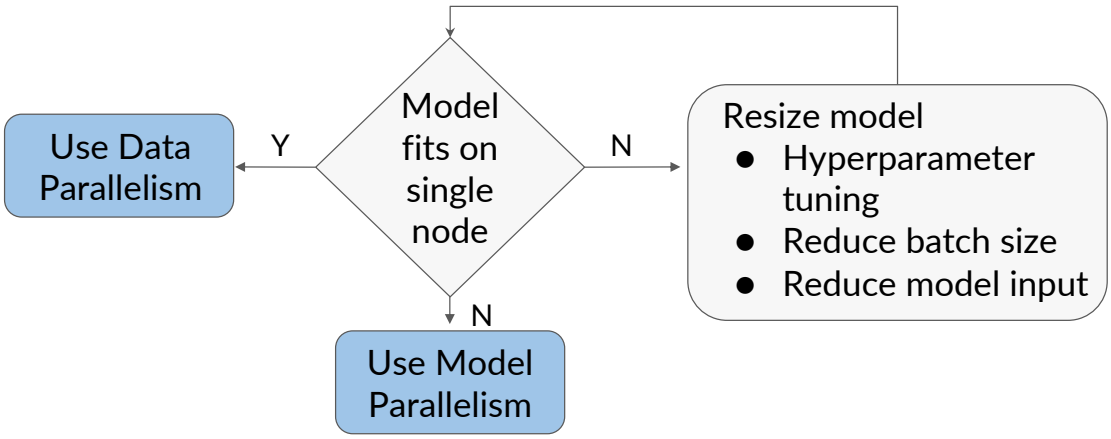
```
from sagemaker.pytorch import PyTorch
estimator = PyTorch(
    entry_point='train.py',
    role=sagemaker.get_execution_role(),
    framework_version='1.6.0',
    py_version='py3',
    instance_count=3,
    instance_type='ml.p3.16xlarge',
    distribution={'smdistributed':{'modelparallel':{enabled': True}}}
)
estimator.fit()
```

Model Parallel Distribution Strategy

Choosing a Distribution Strategy

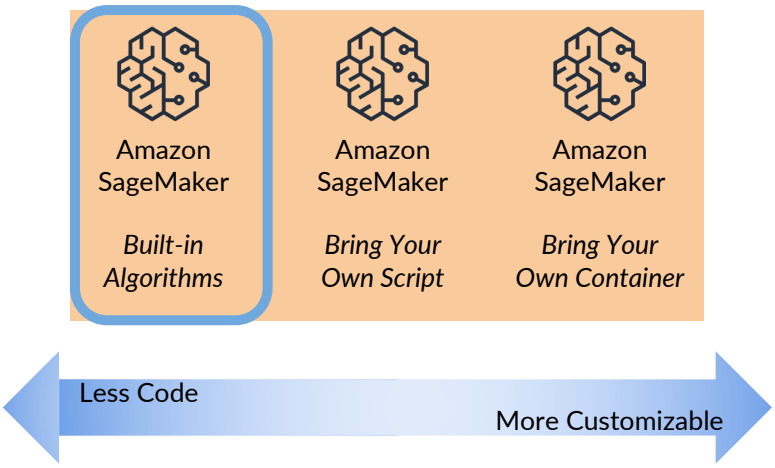


Choosing a Distribution Strategy



Custom Algorithms with Amazon SageMaker

Options on Amazon SageMaker

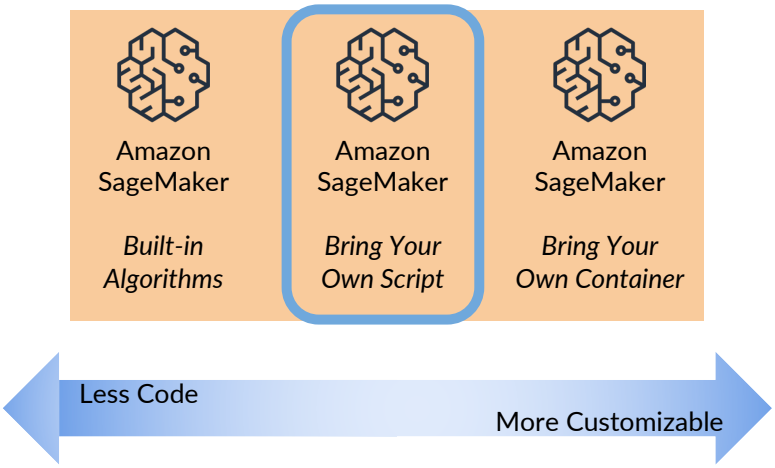


Amazon SageMaker Estimator

```
estimator =
sagemaker.estimator.Estimator(image_uri=image_uri, ...)
estimator.set_hyperparameters(...)
estimator.fit(...)
```

Built-In Algorithms

Options on Amazon SageMaker



Amazon SageMaker Estimator

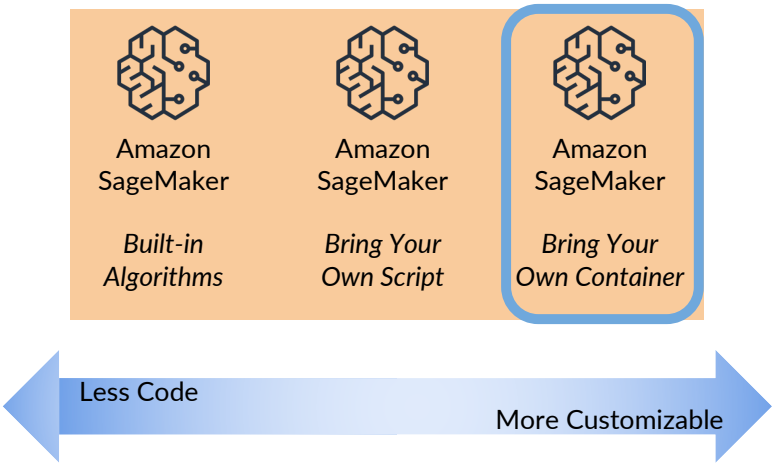
```
estimator =
sagemaker.estimator.Estimator(image_uri=image_uri, ...)
estimator.set_hyperparameters(...)
estimator.fit(...)
```

Built-In Algorithms

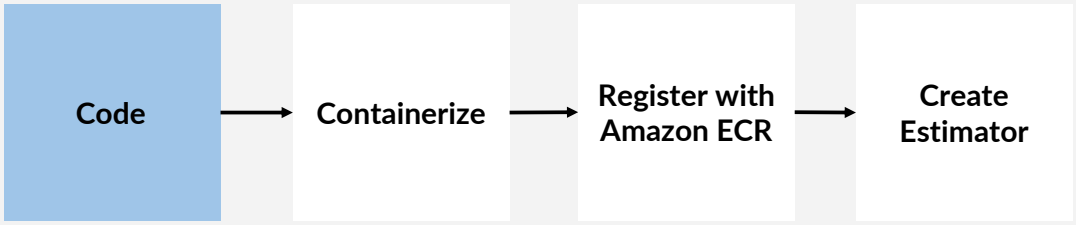
```
from sagemaker.pytorch import PyTorch
pytorch_estimator = PyTorch(
    entry_point='train.py',
    ...
)
```

Script Mode PyTorch Container

Training Options on Amazon SageMaker

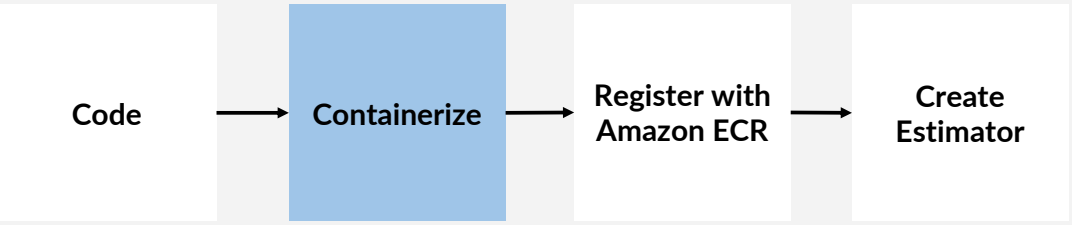


Bring Your Own Container - Steps



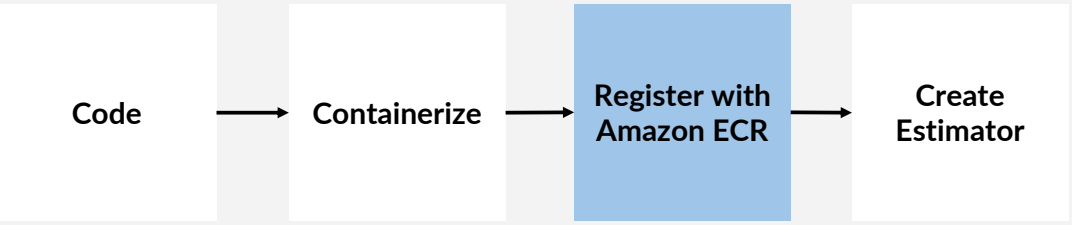
- Algorithm
- Training
- Inference

Bring Your Own Container - Steps



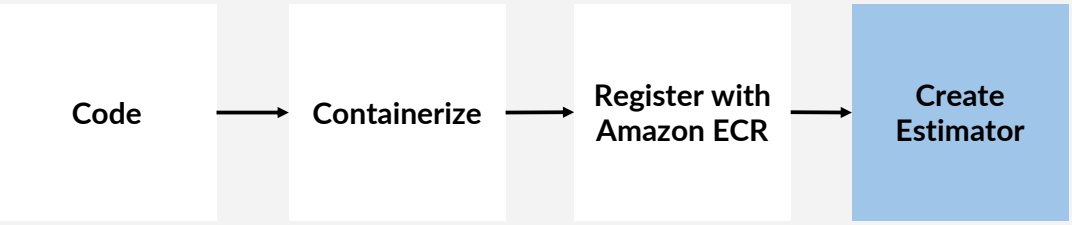
- algorithm_name=tf-custom-container-test
- docker build -t ${algorithm_name} .

Bring Your Own Container - Steps



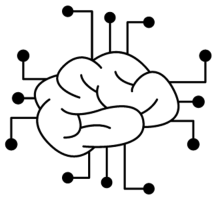
- aws ecr create-repository --repository-name "${algorithm_name}" > /dev/null
- fullname="${account}.dkr.ecr.${region}.amazonaws.com/${algorithm_name}:latest"
- docker push ${fullname}

Bring Your Own Container - Steps



- byoc_image_uri = '{}.dkr.ecr.{}.{}/{}'.format(account_id, region, uri_suffix, ecr_repository + tag)
- estimator = Estimator (image_name=byoc_image_uri, ...)

Summary



- Tune and evaluate a model
- Model tuning
- Tune a BERT-based text classifier
- Model evaluation
- Evaluate a BERT-based text classifier
- TODO: Script Mode?
- TODO: Bring Your Own Container?

Optimize models using Automatic Model Tuning

Introduction

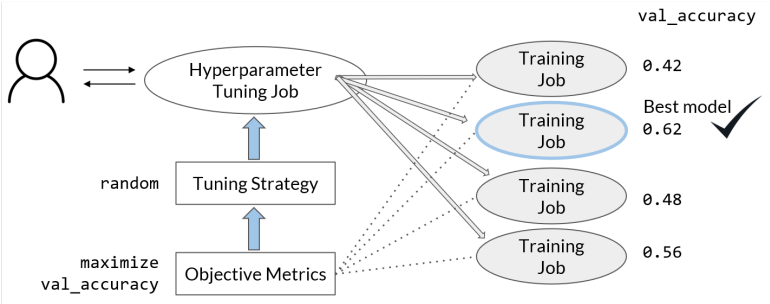
When training ML models, hyperparameter tuning is a step taken to find the best performing training model. In this lab you will apply a random algorithm of Automated Hyperparameter Tuning to train a BERT-based natural language processing (NLP) classifier. The model analyzes customer feedback and classifies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

Table of Contents

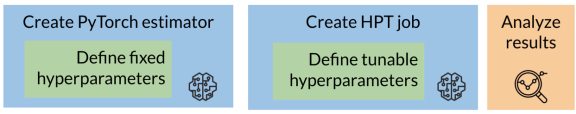
- 1. Configure dataset and Hyperparameter Tuning Job (HTP)
  - 1.1. Configure dataset
    - Exercise 1
  - 1.2. Configure Hyperparameter Tuning Job
  - 1.3. Set up evaluation metrics
- 2. Run tuning job
  - 2.1. Set up the RoBERTa and PyTorch script to run on SageMaker
  - 2.2. Launch the Hyperparameter Tuning Job
    - Exercise 2
    - Exercise 3
  - 2.3. Check Tuning Job status
- 3. Evaluate the results
  - 3.1. Show the best candidate
    - Exercise 4
  - 3.2. Evaluate the best candidate
    - Exercise 5
    - Exercise 6
    - Exercise 7
  - 3.3. Inspect the processed output data

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]:
```
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]:
```
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]:
```
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]:
```
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]:
```
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]:
```
### 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]:
```
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/c21581a44455l1011249t1w593-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-2021-09-16-0
9-27-40-683/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-20
21-09-16-09-27-40-683/output/metrics/confusion_matrix.png to generated/co
nfusion_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 [ ]:

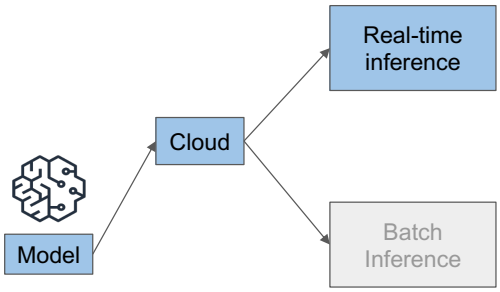
Model Deployment

Overview

Machine Learning Workflow

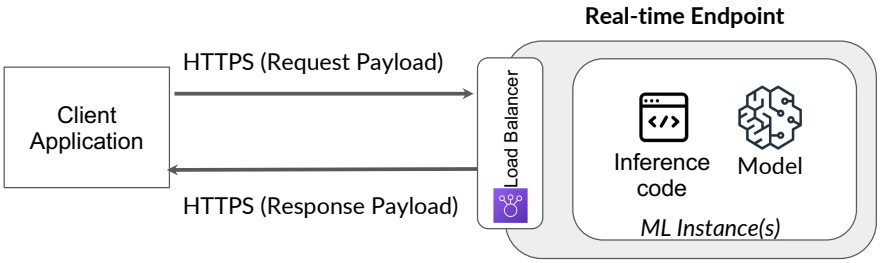
Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Deployment Options



Deployment Options

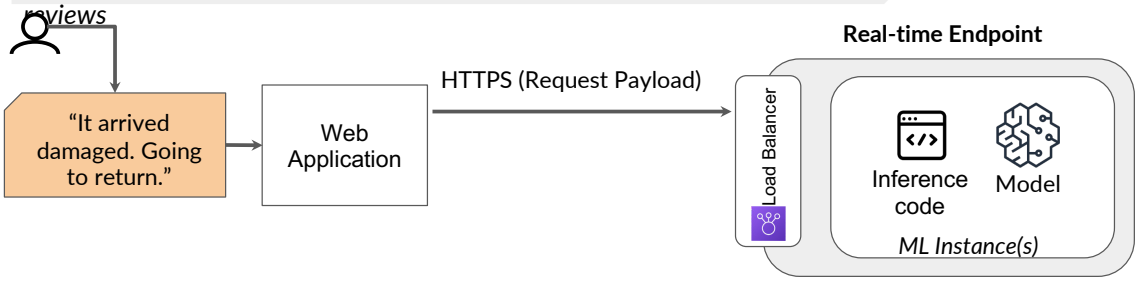
Real-Time Inference



Deployment Options

Real-Time Inference - Product Review Example

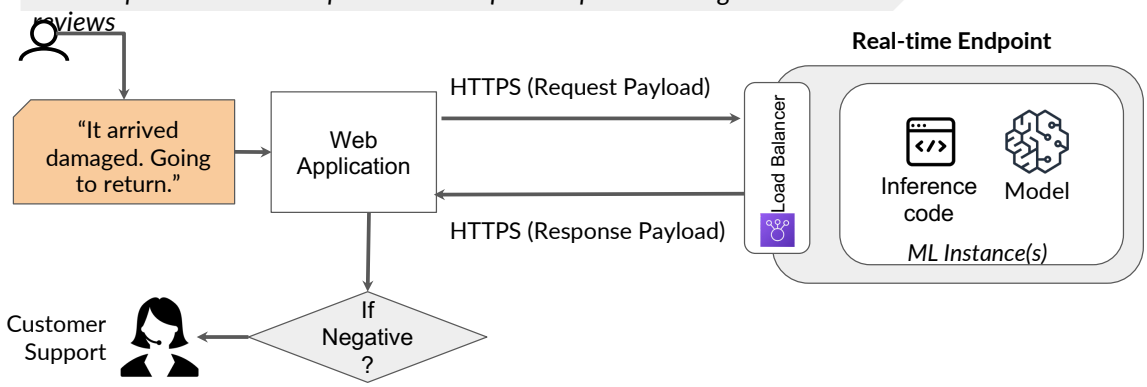
Goal: *Improve customer experience with quick responses to negative reviews*



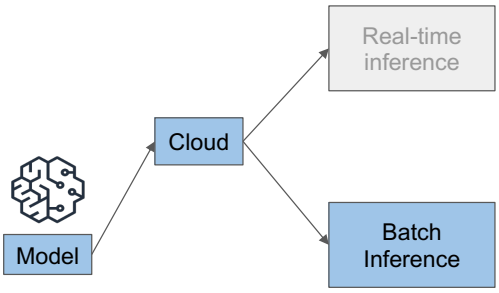
Deployment Options

Real-Time Inference - Product Review Example

Goal: *Improve customer experience with quick responses to negative reviews*



Deployment Options

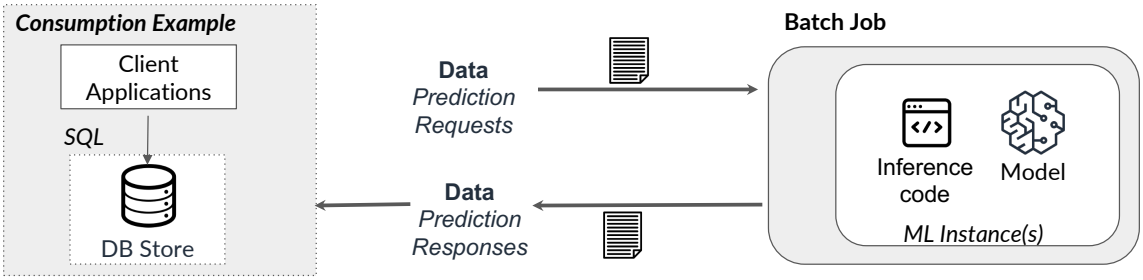


Deployment Options

Batch Inference

Deployment Options

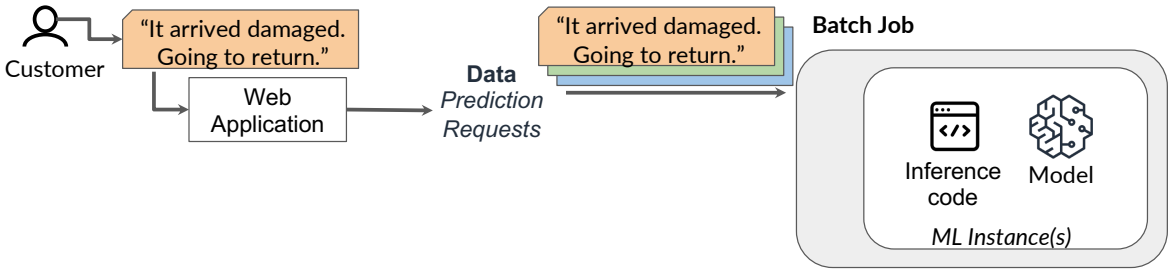
Batch Inference



Deployment Options

Batch Inference - Product Review

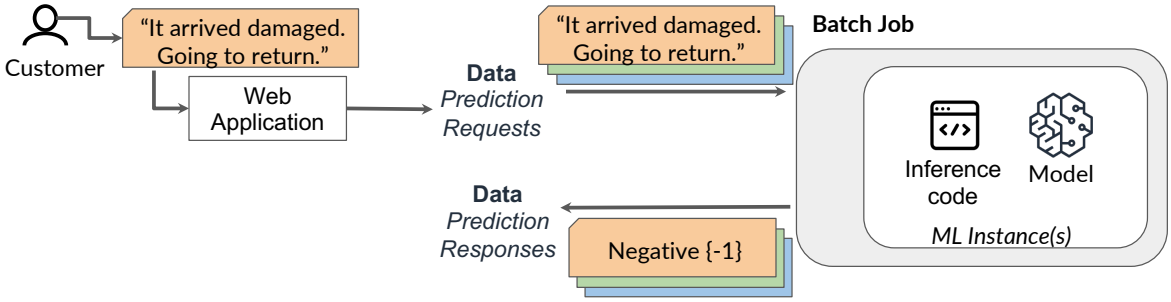
Goal: *Identify vendors with potential quality issues*



Deployment Options

Batch Inference - Product Review

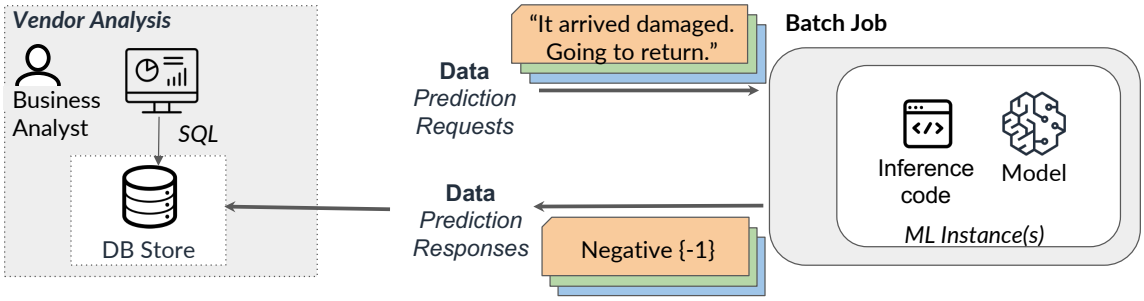
Goal: *Identify vendors with potential quality issues*



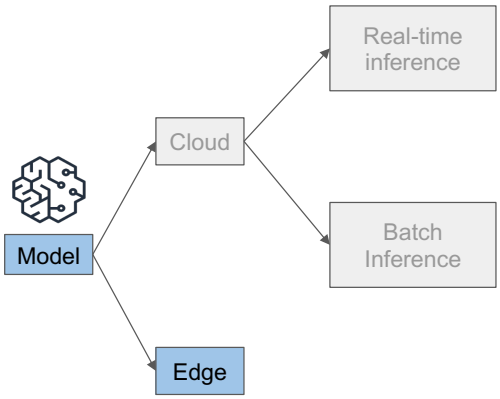
Deployment Options

Batch Inference - Product Review

Goal: *Identify vendors with potential quality issues*

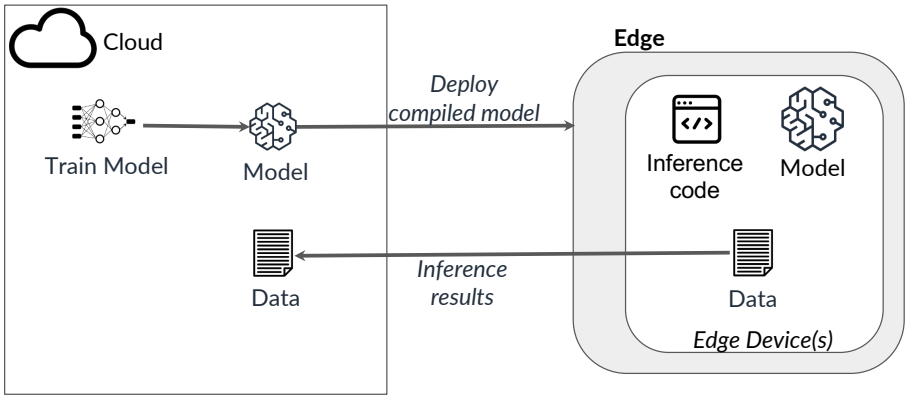


Deployment Options



Deployment Options

Edge



Deployment Options

Choose the deployment option that best fits the use case

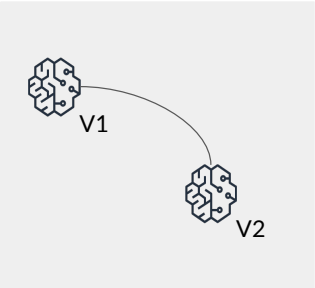
	Real-Time Inference	**Batch Inference**	**Edge**
When to use	Low latency real-time predictions (Ex. Interactive Recommenders)	Batch request & response prediction is acceptable for your use case (Ex. Forecasting)	Models need to deployed to edge devices (Ex. Limited connectivity, Internet of Things)
Cost	Persistent endpoint - pay for resources while endpoint is running	Transient environments - pay for resources for the duration of the batch job	Varies

Model Deployment

Deployment Strategies

Deployment Strategies

Strategies to deploy new and updated models



Goals:

- Minimize risk
- Minimize downtime
- Measure model performance

Deployment Strategies

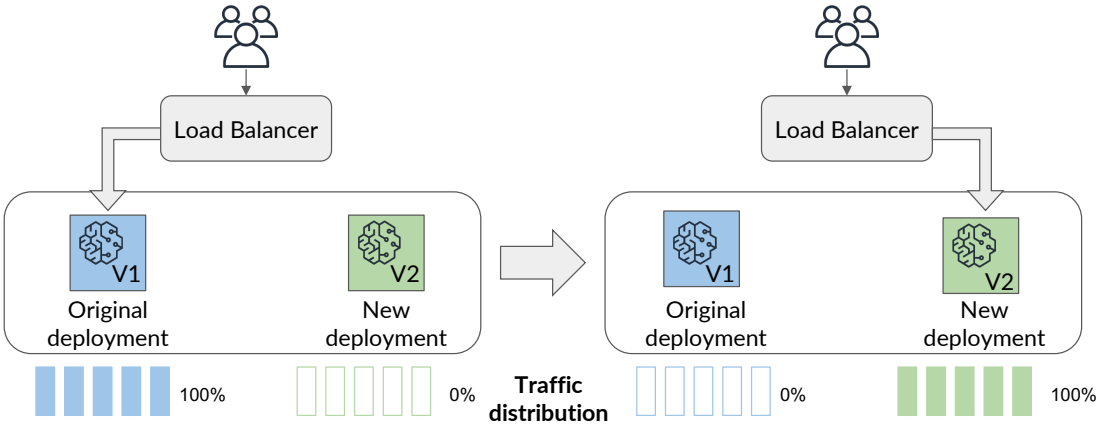
Common Strategies to deploy new and updated models

Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

- Swap prediction request traffic
- Easy Rollback

Deployment Strategies

Blue/Green: Shift all traffic to the new model



Deployment Strategies

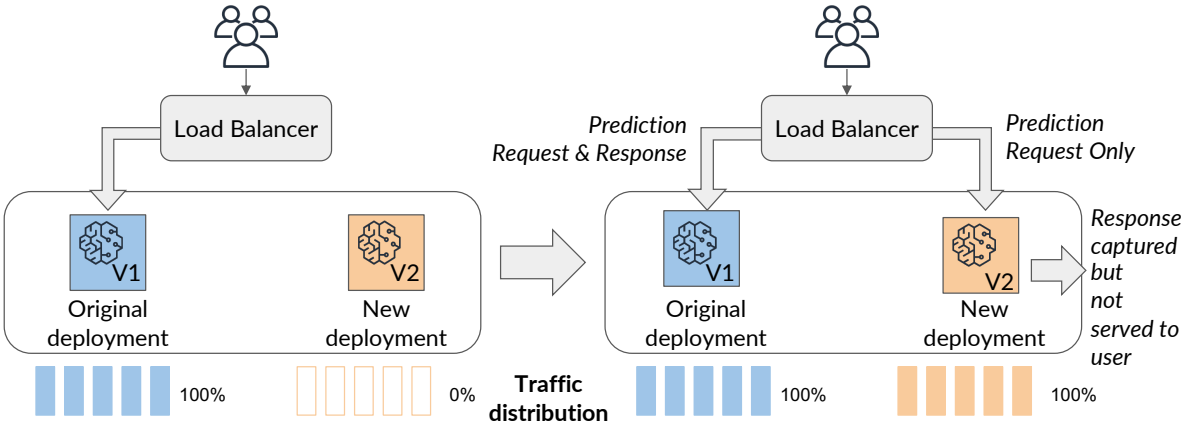
Common Strategies to deploy new and updated models

Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

- Parallel prediction request traffic
- Validate new version without impact

Deployment Strategies

Shadow/Challenger: Run multiple versions in parallel with one serving live traffic



Deployment Strategies

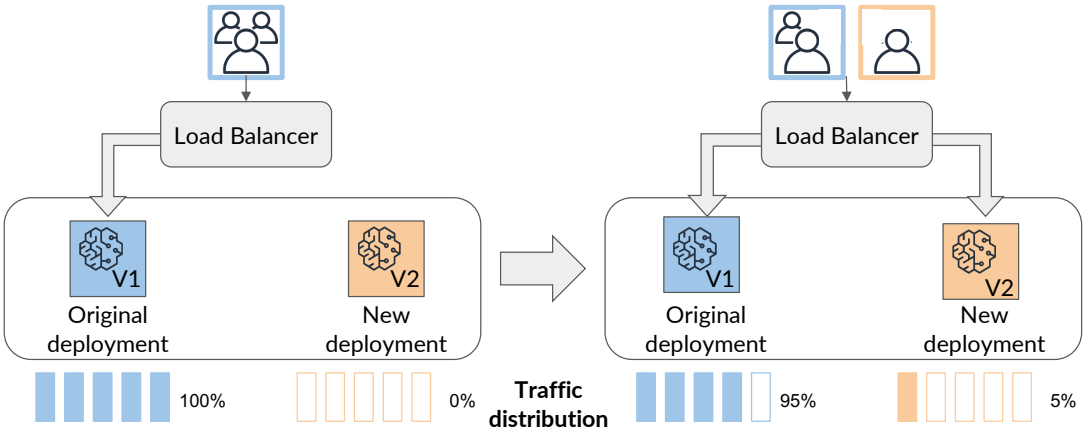
Common Strategies to deploy new and updated models

Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

- Split traffic
- Target smaller specific users/groups
- Shorter validation cycles
- Minimize risk of low performing model

Deployment Strategies

Canary: Split traffic to compare model versions with target groups/users



Deployment Strategies

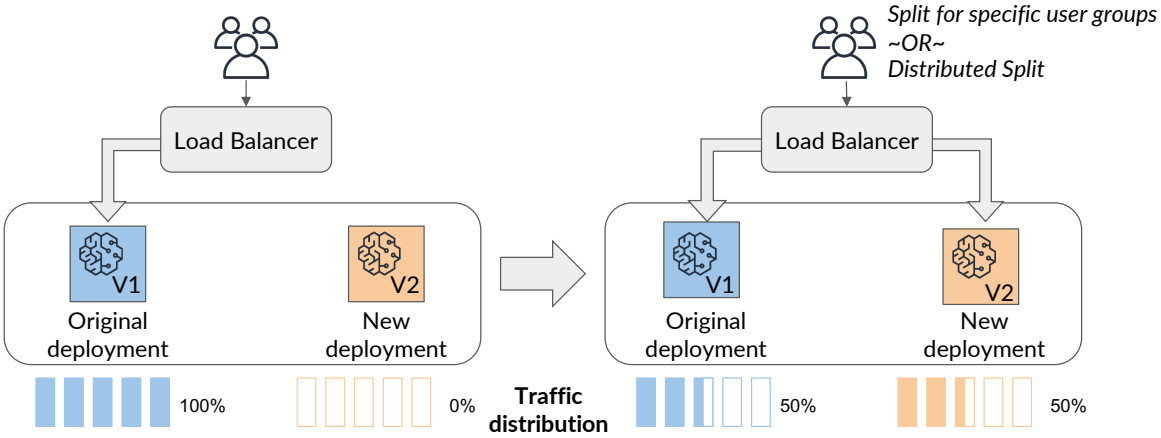
Common Strategies to deploy new and updated models

Blue/Green	Shadow/ Challenger	Canary	A/B	Multi-Armed Bandits

- Split traffic
- Target larger users/groups ~OR~ Distribute % of traffic
- Longer validation cycles
- Minimize risk of low performing model

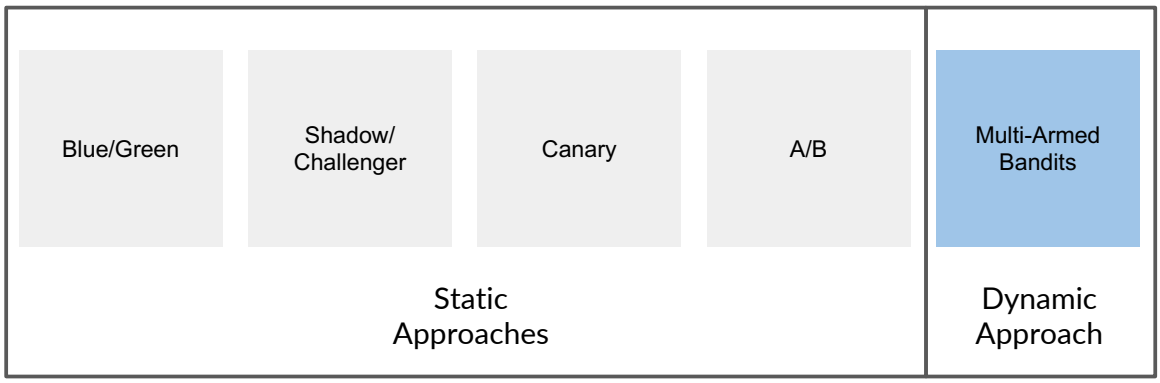
Deployment Strategies

A/B: Split traffic to compare model versions



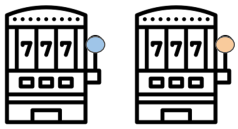
Deployment Strategies

Common Strategies to deploy new and updated models



Deployment Strategies

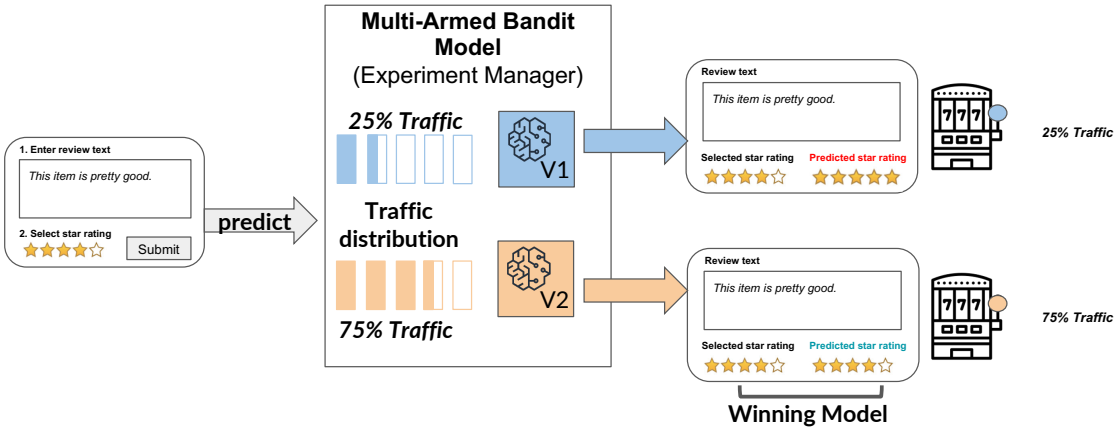
Multi-Armed Bandits (MABs)



- **Dynamic testing method for model version using Reinforcement Learning**
- **Exploit & Explore**
  - **Exploit:** Reward the winning model with more traffic
  - **Explore:** Continue to send traffic to the non-winning model(s) in case behavior changes

Deployment Strategies

Multi-Armed Bandits: Dynamically shift traffic to the winning model



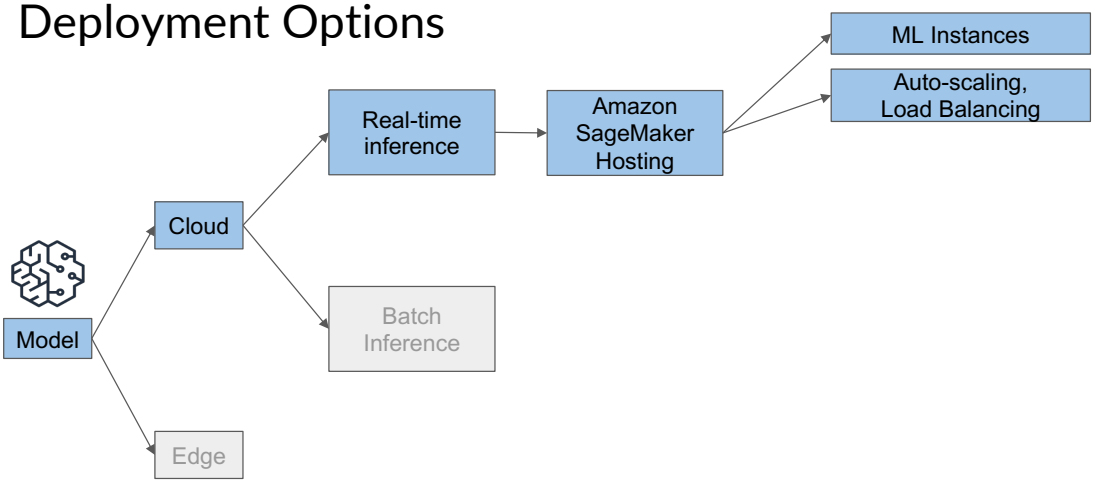
Amazon SageMaker Hosting

Real-Time Inference

Machine Learning Workflow

Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

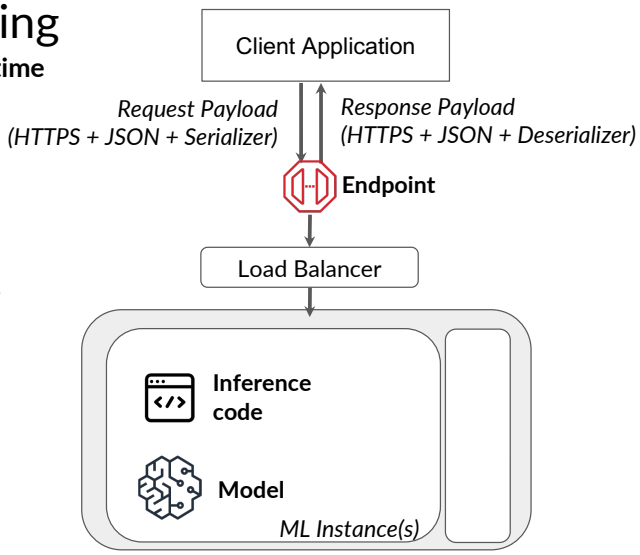
Deployment Options



Amazon SageMaker Hosting

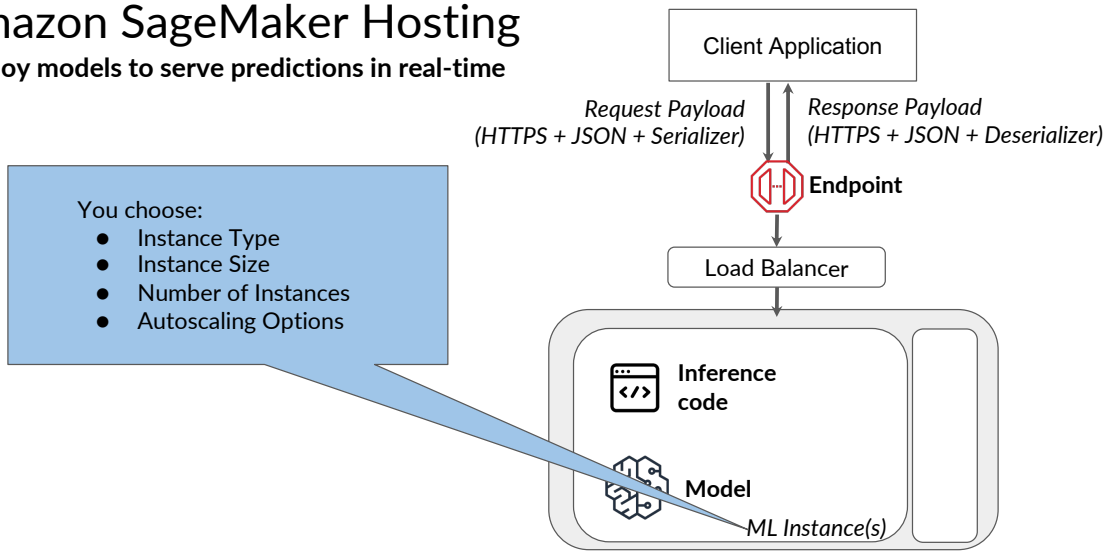
Deploy models to serve predictions in real-time

- Optimized for **low latency** of model predictions
- **Example:** As product reviews are coming in through online channels, you want to predict the sentiment for immediate action



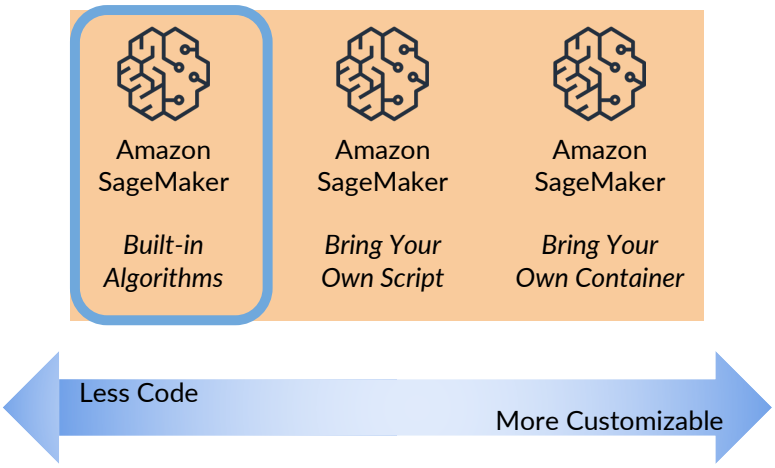
Amazon SageMaker Hosting

Deploy models to serve predictions in real-time



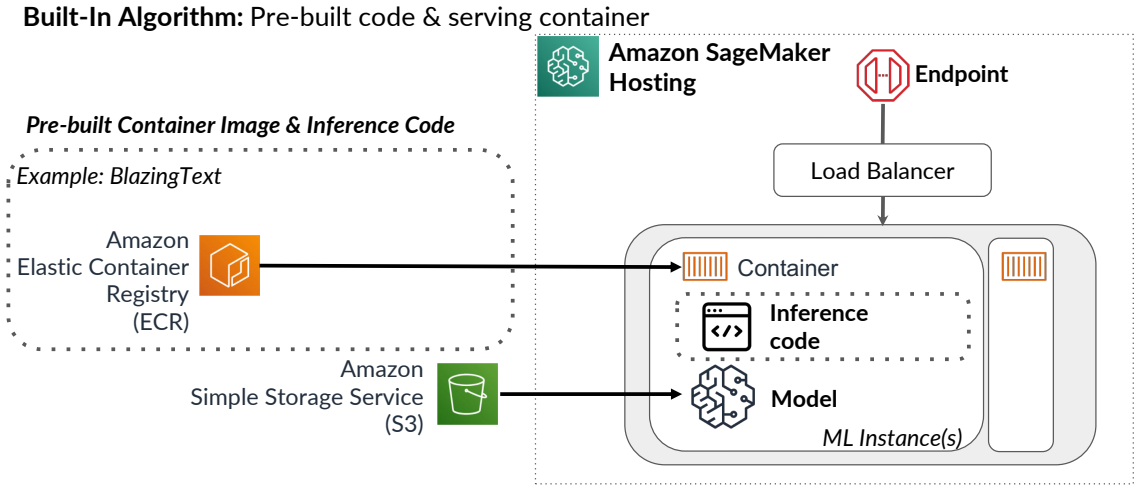
Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time



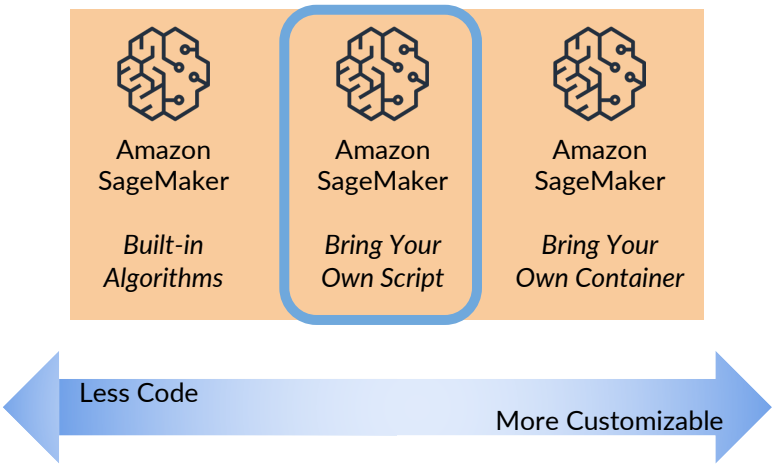
Amazon SageMaker Hosting

Built-In Algorithm: Pre-built code & serving container



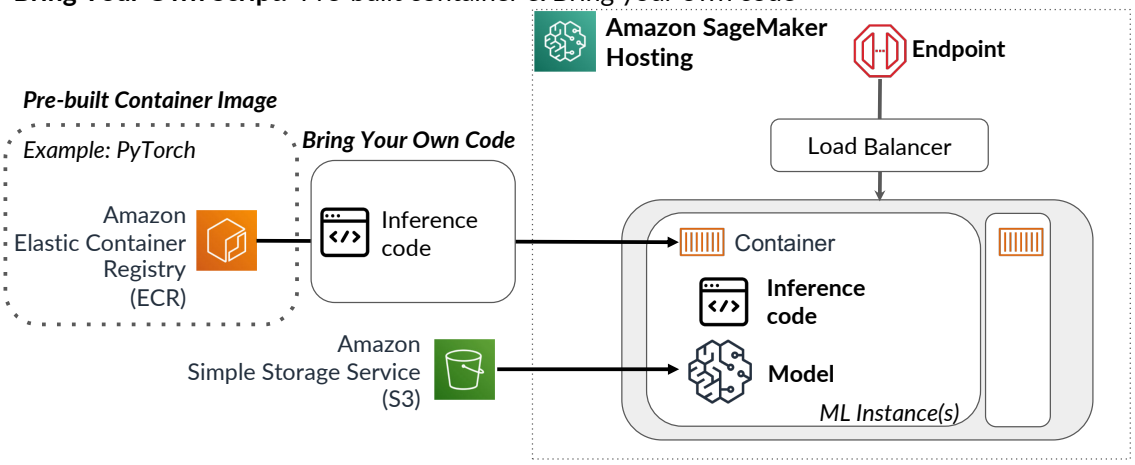
Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time



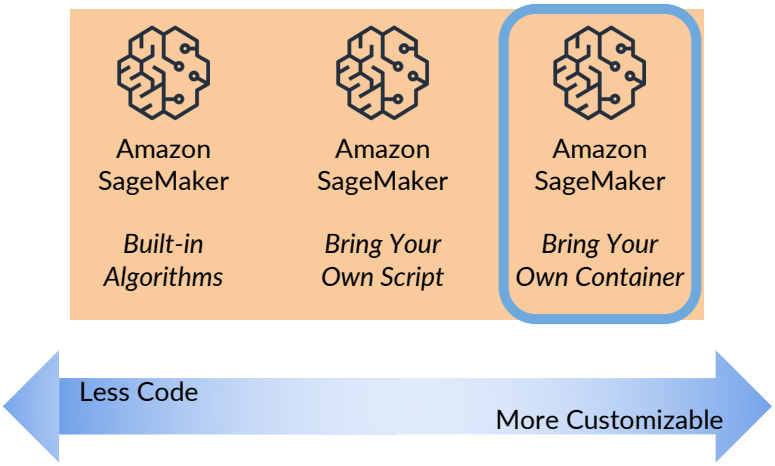
Amazon SageMaker Hosting

Bring Your Own Script: Pre-built container & Bring your own code



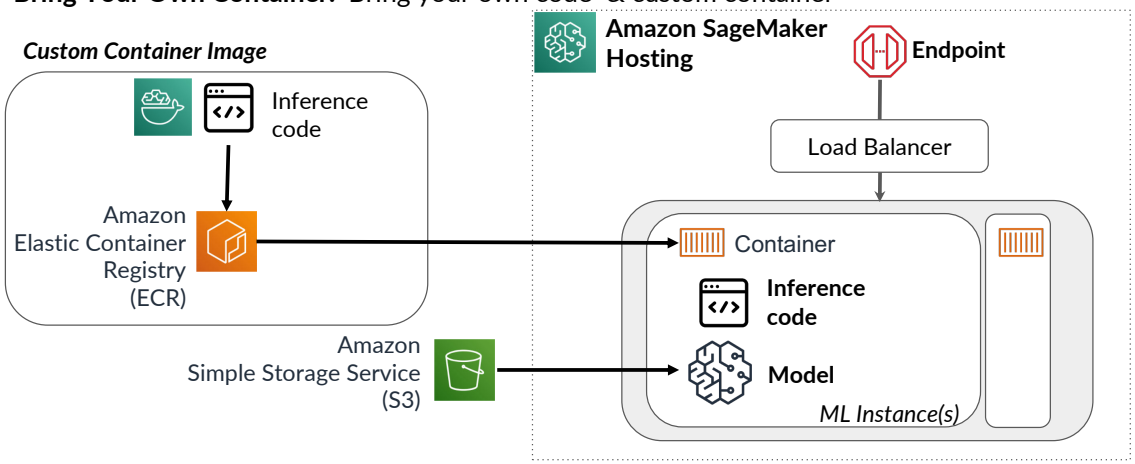
Amazon SageMaker Hosting

Options to deploy models to serve predictions in real-time



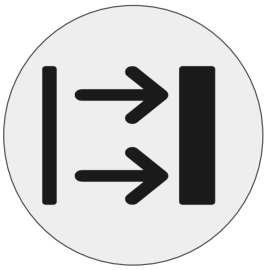
Amazon SageMaker Hosting

Bring Your Own Container: Bring your own code & custom container



Amazon SageMaker Hosting

Autoscaling Endpoints

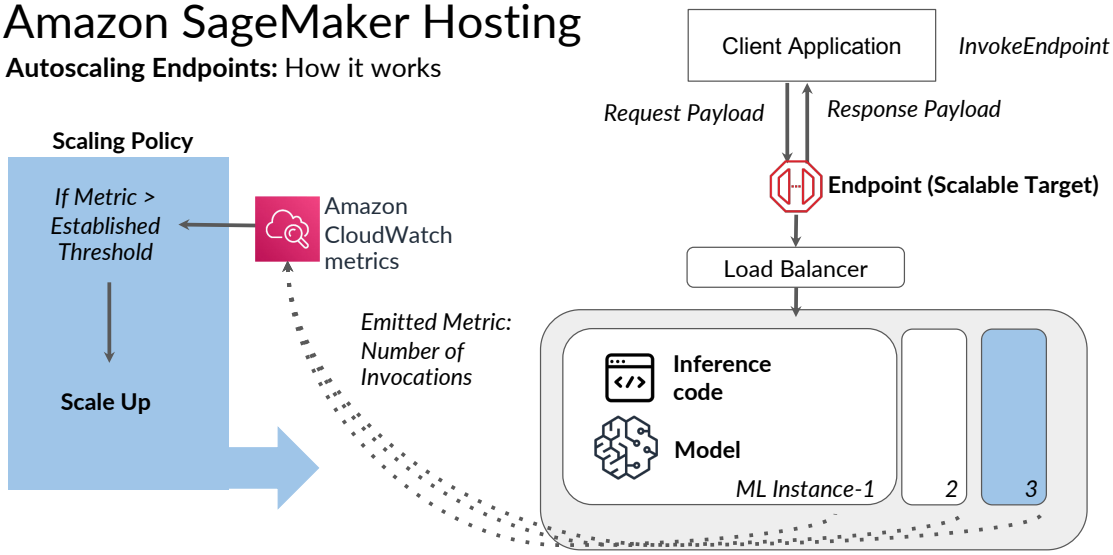


Why?

- Ensure you can meet the demands of your workload
- Cost optimization

Amazon SageMaker Hosting

Autoscaling Endpoints: How it works

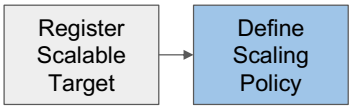


Autoscale Amazon SageMaker Endpoints

Register Scalable Target

```
autoscale.register_scalable_target(
          ServiceNamespace="sagemaker",
    ResourceId="endpoint/" + endpoint_name,
    ScalableDimension="sagemaker:variant:DesiredInstanceCount",
    MinCapacity=1,
    MaxCapacity=2,
    RoleARN=role,
    SuspendedState={
        "DynamicScalingInSuspended": False,
        "DynamicScalingOutSuspended": False,
        "ScheduledScalingSuspended": False,
    })
```

Autoscale Amazon SageMaker Endpoints



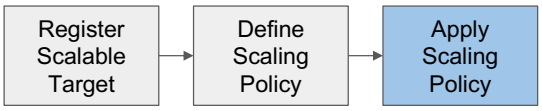

```
scaling_policy = {
        "TargetValue": 2.0,
        "PredefinedMetricSpecification": {
        "PredefinedMetricType": "SageMakerVariantInvocationsPerInstance",
        },
        "ScaleOutCooldown": 60,
        "ScaleInCooldown": 300,
        },
}
```

Scaling Metric

Wait time, in seconds, before beginning another **scale out** activity after last one completes

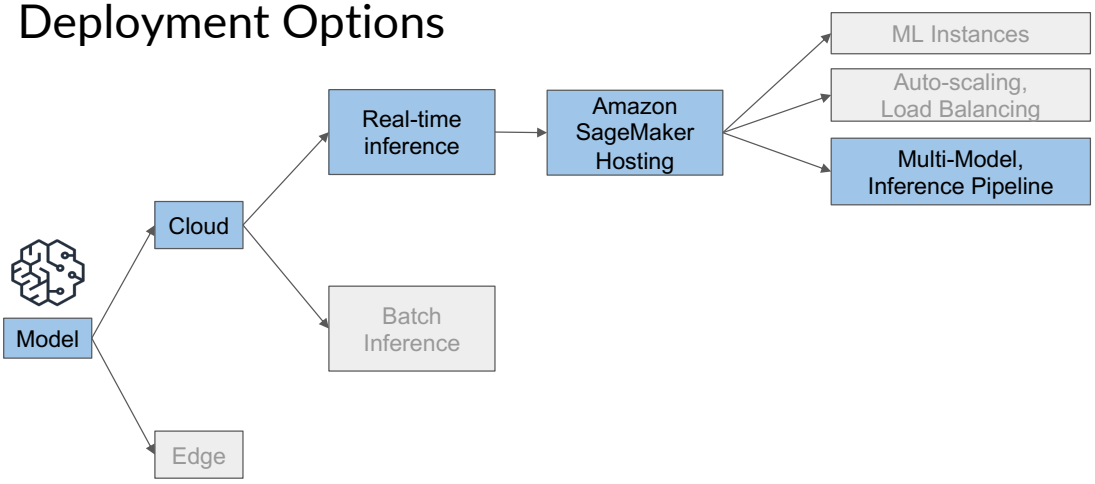
Wait time, in seconds, before beginning another **scale in** activity after last one completes

Autoscale Amazon SageMaker Endpoints




```
autoscale.put_scaling_policy(
        PolicyName=...,
        ServiceNamespace="sagemaker",
        ResourceId="endpoint/" + endpoint_name,
        ScalableDimension="sagemaker:variant:DesiredInstanceCount",
        PolicyType="TargetTrackingScaling",
        TargetTrackingScalingPolicyConfiguration=scaling_policy)
```

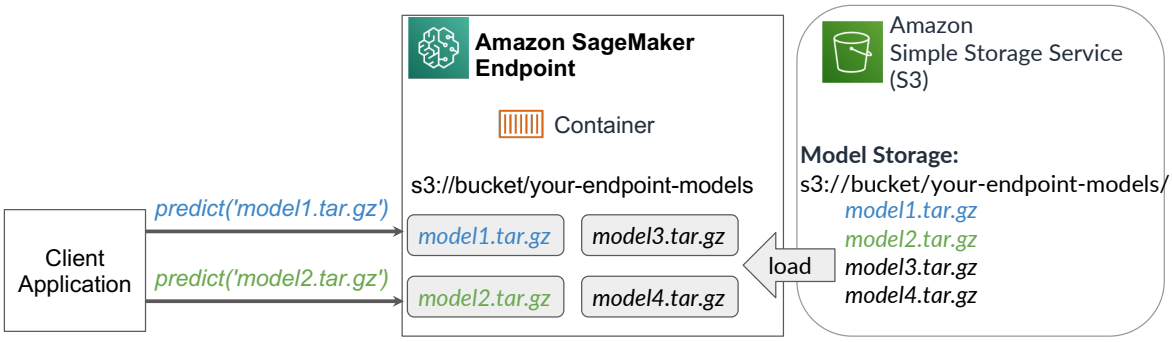

Deployment Options



Advanced Deployment Options

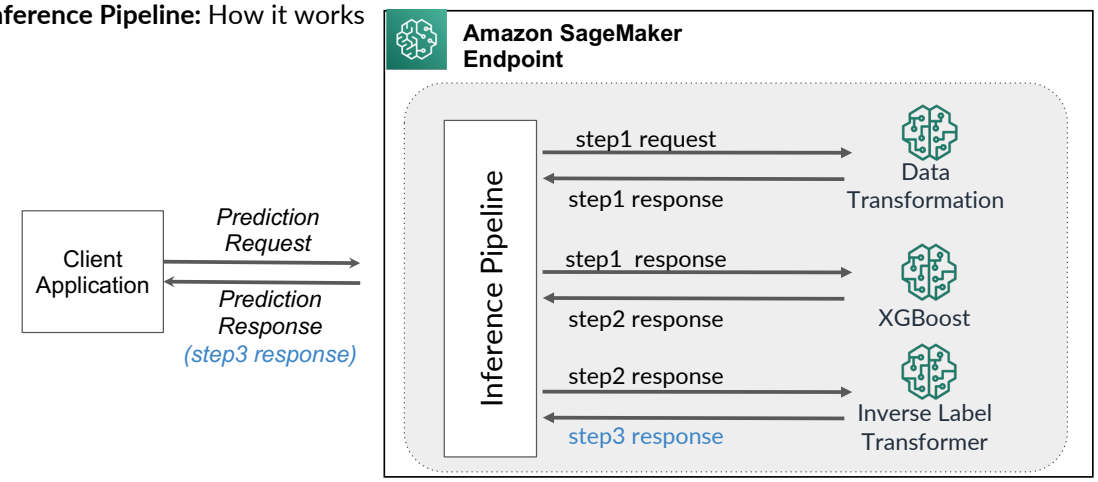
Multi-Model Endpoints: How it works

Deploy Multiple Models to a Single Endpoint



Advanced Deployment Options

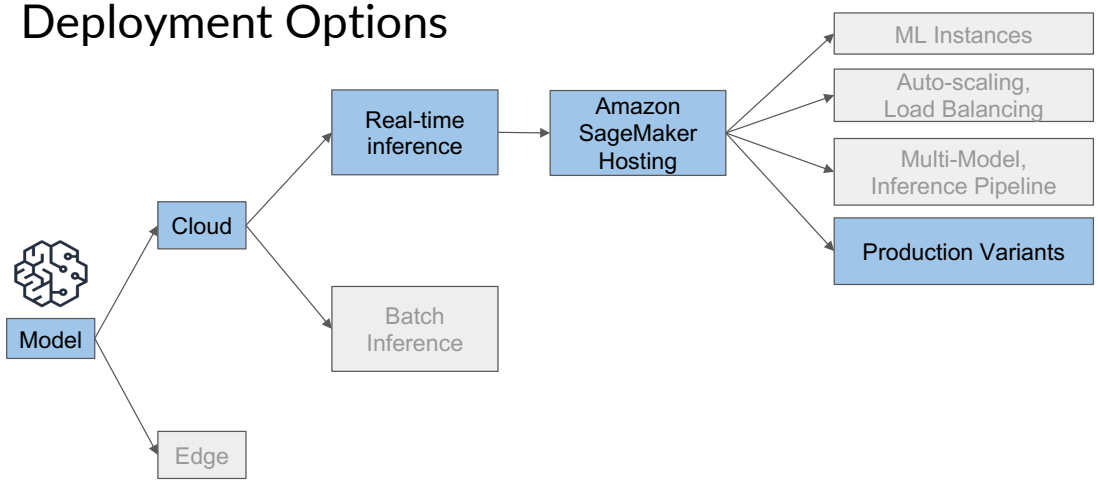
Inference Pipeline: How it works



Amazon SageMaker Hosting

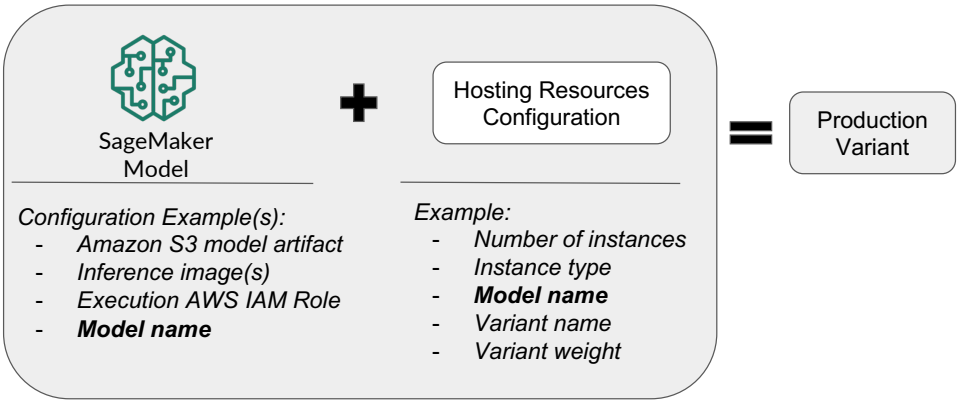
Real-Time Inference
Production Variants

Deployment Options



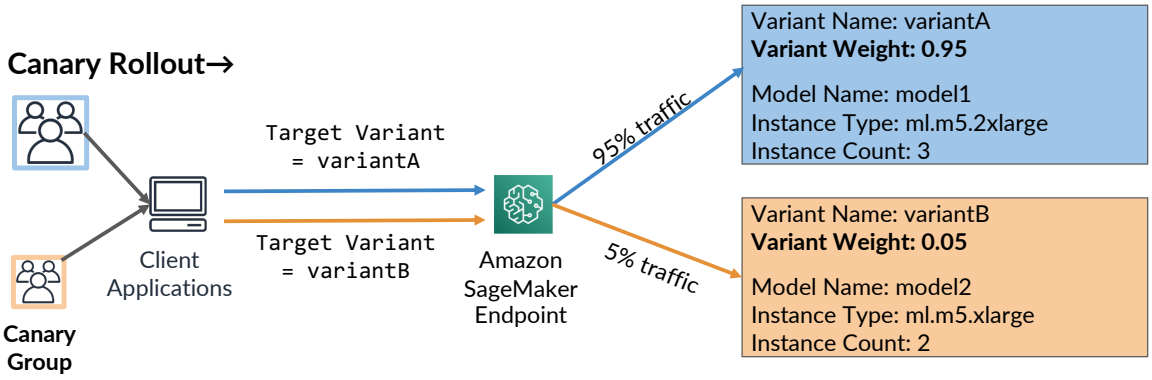
Amazon SageMaker Production Variants

What is a Production Variant?



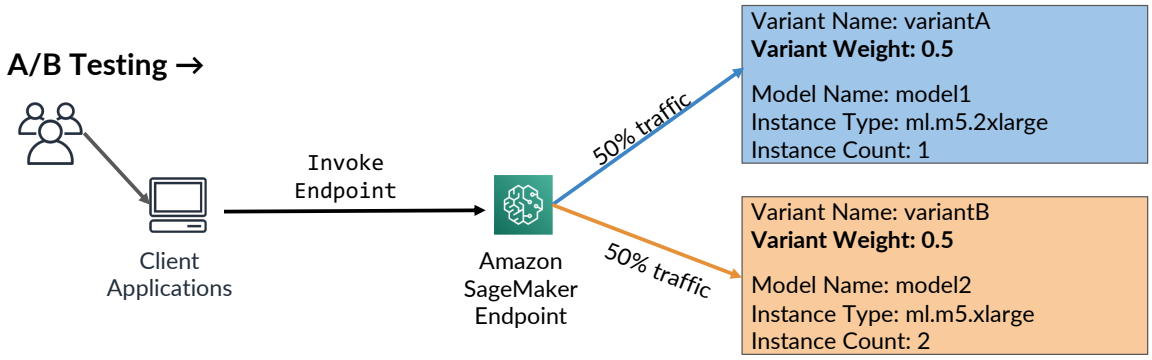
Amazon SageMaker Production Variants

Using Production Variants for a Canary Rollout



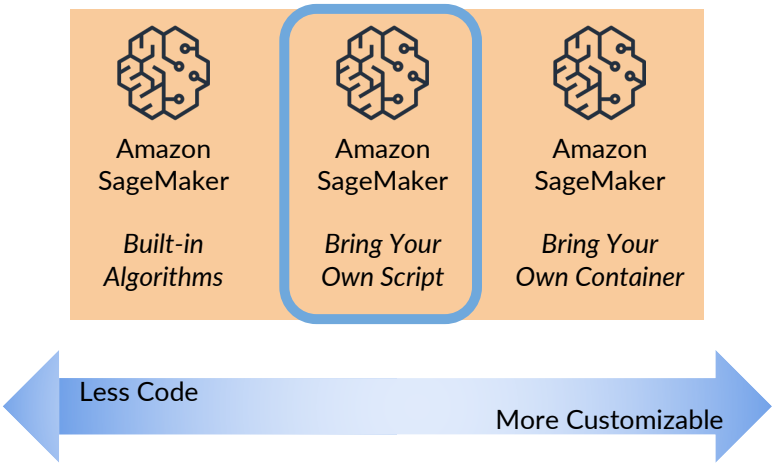
Amazon SageMaker Production Variants

Using Production Variants for A/B Testing



Amazon SageMaker Hosting

Using Production Variants for A/B Testing with Bring-Your-Own Script



Using Production Variants for A/B Testing

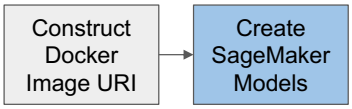
A/B Testing with PyTorch Bring-Your-Own Script

Construct Docker Image URI

```
import sagemaker

inference_image_uri = sagemaker.image_uris.retrieve(
     framework=..., # PyTorch, TensorFlow, etc...
     version='1.6.0',
     instance_type='ml.m5.xlarge',
     py_version='py3',
     image_scope='inference'
)
```

Using Production Variants for A/B Testing

A/B Testing with PyTorch Bring-Your-Own Script




```
sm.create_model(
          name=model_name_a,
          ...
)
```

```
sm.create_model(
          name=model_name_b,
          ...
)
```

SageMaker Model A

SageMaker Model B

Using Production Variants for A/B Testing

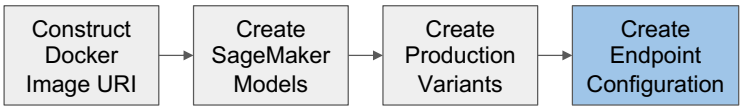
A/B Testing with PyTorch Bring-Your-Own Script

```
from sagemaker.session \
    import production_variant

variantA = production_variant(
          model_name=...,
     instance_type=...,
          initial_instance_count=1,
          variant_name='VariantA',
          initial_weight=50,
)
```

```
from sagemaker.session \
     import production_variant

variantB = production_variant(
          model_name=...,
     instance_type=...,
          initial_instance_count=1,
          variant_name='VariantB',
          initial_weight=50,
)
```


Using Production Variants for A/B Testing

A/B Testing with PyTorch Bring-Your-Own Script




```
endpoint_config = sm.create_endpoint_config(
          EndpointConfigName=...,
     ProductionVariants=[variantA, variantB]
)
```

Using Production Variants for A/B Testing

A/B Testing with PyTorch Bring-Your-Own Script

```
endpoint_response = sm.create_endpoint(EndpointName=...,
                                       EndpointConfigName=...)
```

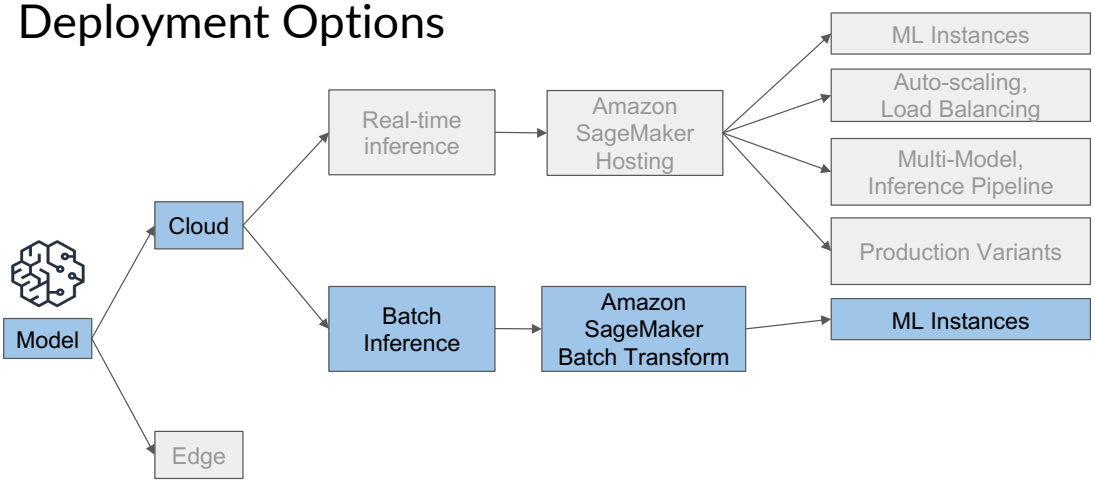

Amazon SageMaker Batch Transform

Batch Inference

Machine Learning Workflow

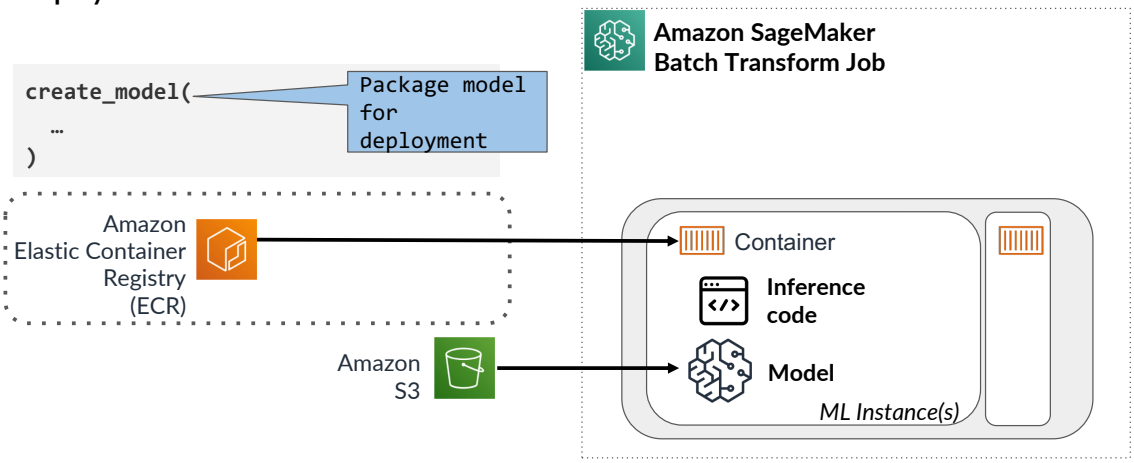
Ingest & Analyze	Prepare & Transform	Train & Tune	Deploy & Manage
Data exploration	Feature engineering	Automated ML	Model deployment
Bias detection	Feature store	Model train and tune	Automated pipelines
Amazon S3 & Amazon Athena	Amazon SageMaker Data Wrangler	Amazon SageMaker Autopilot	Amazon SageMaker Endpoints
AWS Glue	Amazon SageMaker Processing Jobs	Amazon SageMaker Training & Debugger	Amazon SageMaker Batch Transform
Amazon SageMaker Data Wrangler & Clarify	Amazon SageMaker Feature Store	Amazon SageMaker Hyperparameter Tuning	Amazon SageMaker Pipelines

Deployment Options



Amazon SageMaker Batch Transform

Deploy Model For Batch Inference

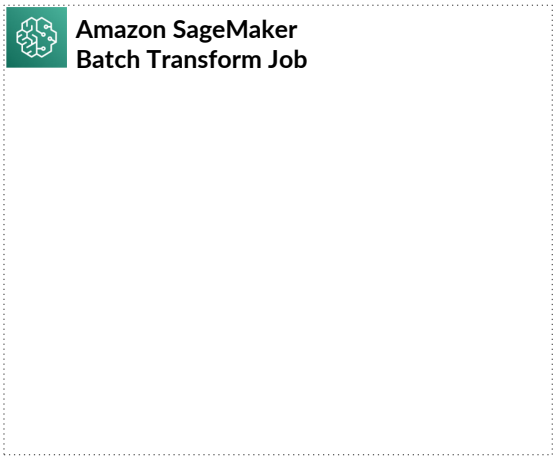


Amazon SageMaker Batch Transform

Run Batch Transform Job For Batch Inference

```
sm_transformer = Transformer(
    ...
)
```

Configure Batch Transform Job

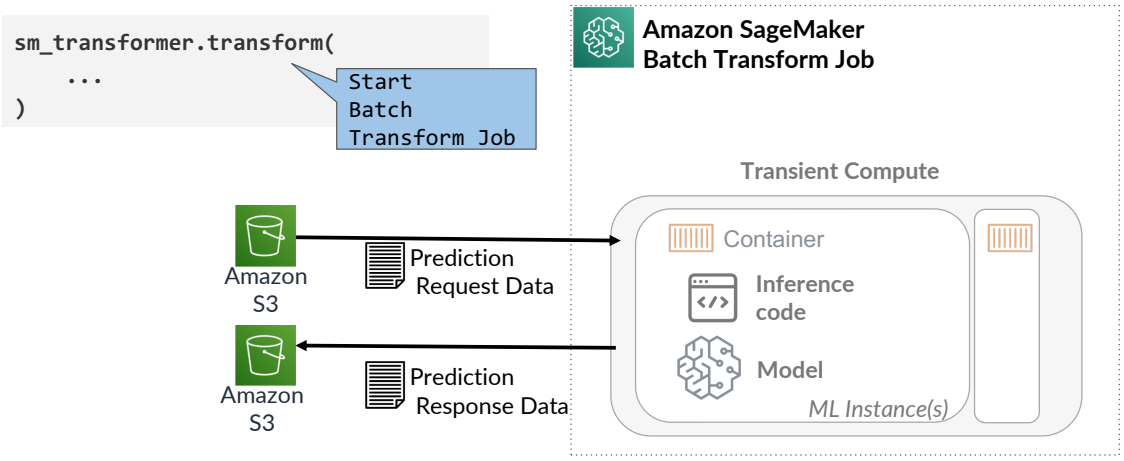


Identify Configuration →

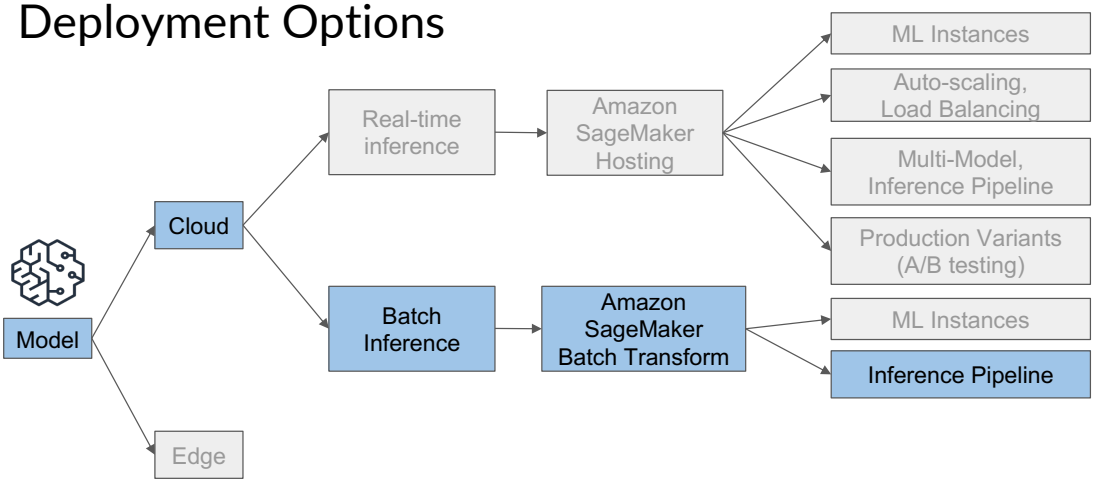
- Instance type
- Instance count
- Model name
- S3 output path
- …

Amazon SageMaker Batch Transform

Run Batch Transform Job For Batch Inference

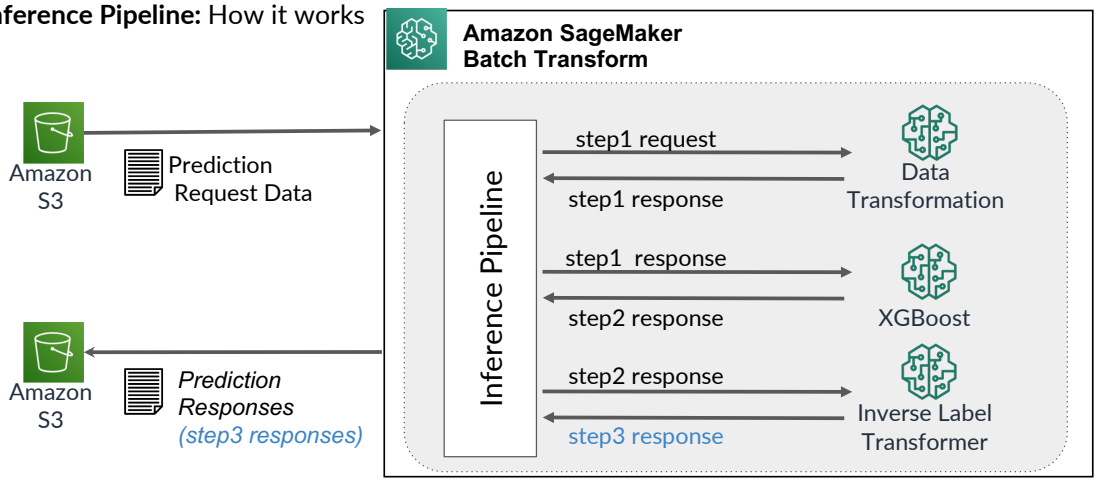


Deployment Options



Advanced Deployment Options

Inference Pipeline: How it works

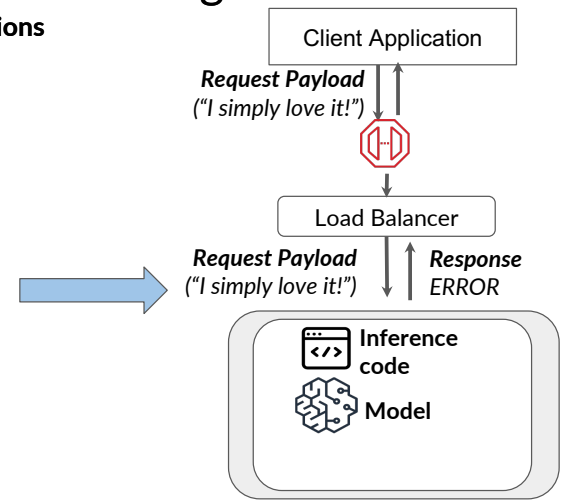


Model Integration

Amazon SageMaker Model Integration

Integrating Models with ML Applications

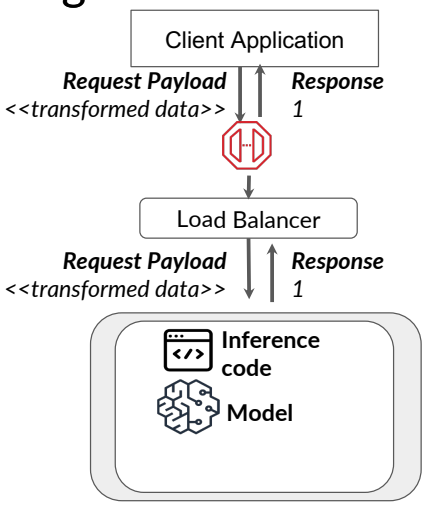
Need to apply the same data transformations used during training



Amazon SageMaker Model Integration

Prepare Data for Inference in Client Application

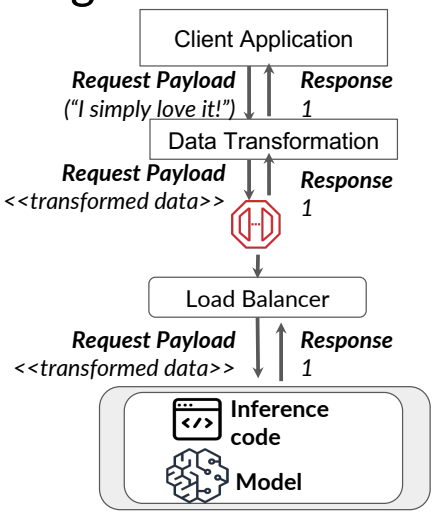
- **Implement data transformations in Client Application**
  - **Challenge:** Difficult to scale & manage
  - **Consideration:** Response may need to be transformed (1 = Positive)



Amazon SageMaker Model Integration

Prepare Data for Inference in Client Application

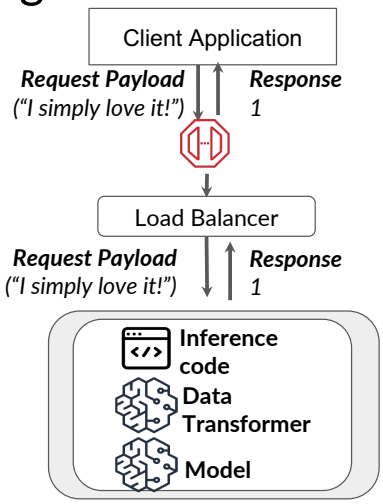
- **Implement transformation code before calling hosted model**
  - **Challenge:** Need to ensure transformation code stays in sync with training code



Amazon SageMaker Model Integration

Prepare Data as Part of an Inference Pipeline

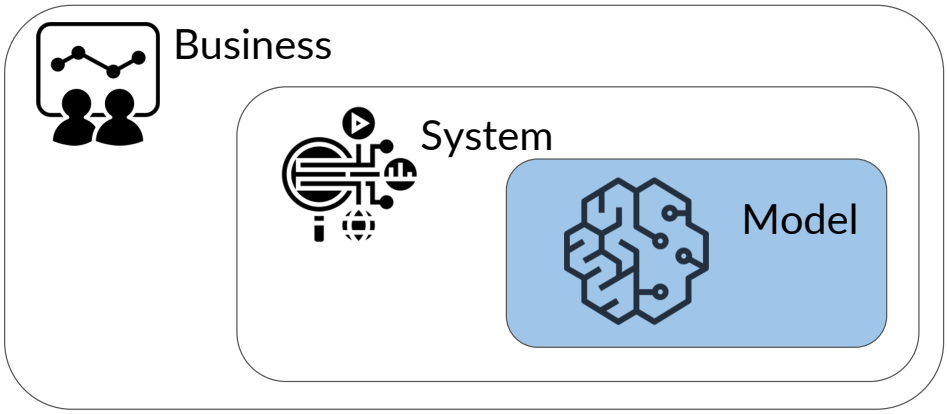
- **Implement data transformations in Inference Pipeline**
  - **Benefit:** Keep training & inference code in sync
  - **Consideration:** Additional Data Transformer for response may need to be transformed (1 = Positive)



Monitoring ML Workloads

Monitoring Machine Learning Workloads

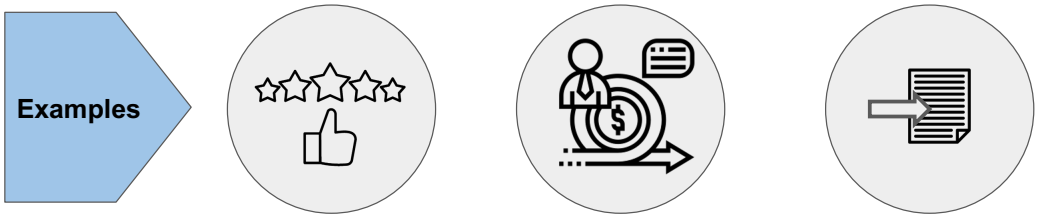
Considerations



Monitoring Machine Learning Workloads

Model Monitoring

Why? Models degrade over time



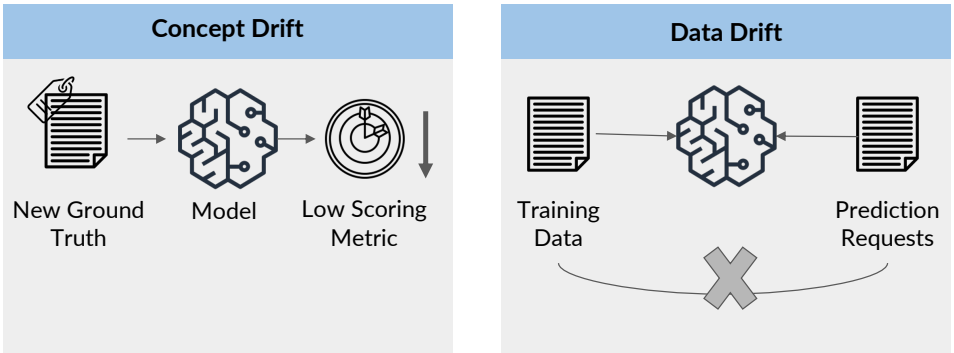
Customer behavior change
Ex. Product change, Demand change

Changing business environment
Ex. New products

Changing data pipeline
Ex. Feature data suddenly missing

Monitoring Machine Learning Workloads

Model Monitoring



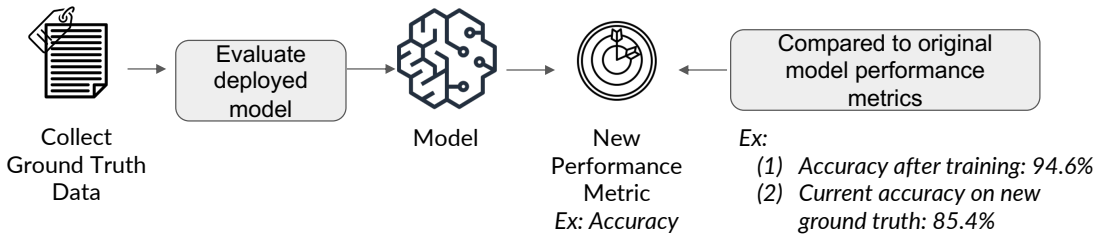
Monitoring Machine Learning Workloads

Concept Drift

What causes **concept drift?**

- Environment changes that impact the context of the predicted target

Methods to detect:



Monitoring Machine Learning Workloads

Data Drift

What causes **Data Drift?**

- Changes in the model input data

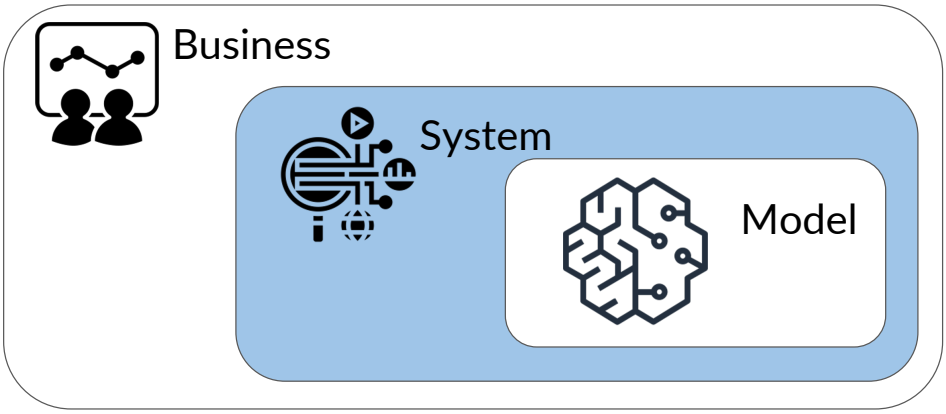
Methods to detect:

Example: Deequ - Open Source Library

- **Data Profiling:** Gather statistics about each feature used to train the model
- **Establish Constraints:** Boundaries on normal/expected data
- **Detect Data Anomalies:** Understand when prediction data violates constraints

Monitoring Machine Learning Workloads

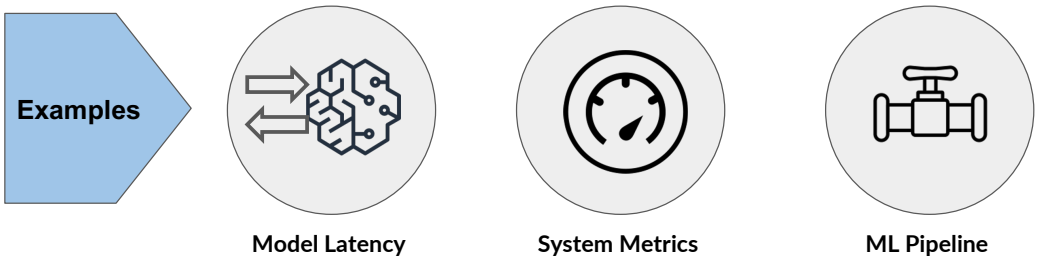
Considerations



Monitoring Machine Learning Workloads

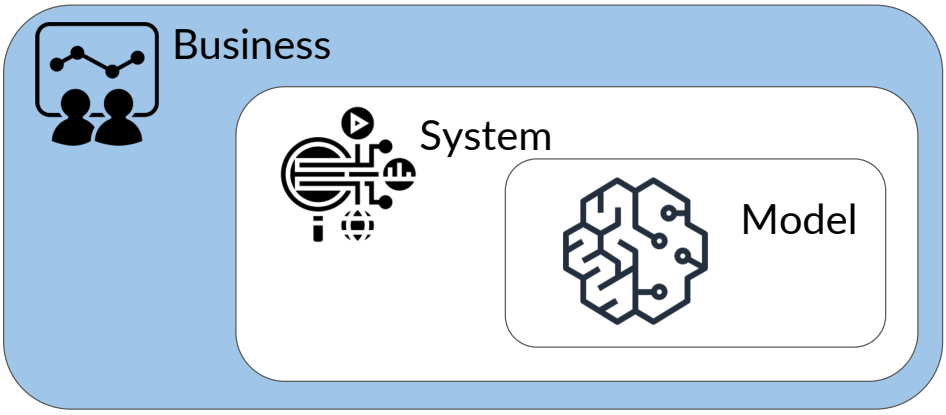
System Monitoring

Why? Ensure your model and supporting resources are functioning as expected



Monitoring Machine Learning Workloads

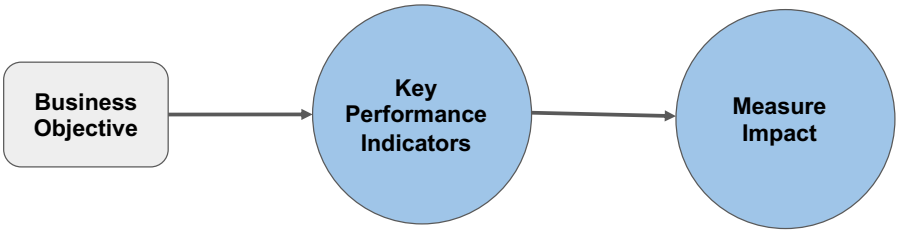
Considerations



Monitoring Machine Learning Workloads

Monitoring Impact on Business Objectives

Why? Ensure your model has impact on the business objective



Model Monitoring

Using Amazon SageMaker Model Monitor

Amazon SageMaker Model Monitor

Monitor Types

Data Drift	*Concept Drift*	*Concept Drift*	*Data Drift*
Data Quality	**Model Quality**	**Statistical Bias Drift**	**Feature Attribution Drift**
Monitor drift in data quality	Monitor drift in model quality metrics	Monitor statistical bias drift in model predictions	Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor

Data Drift

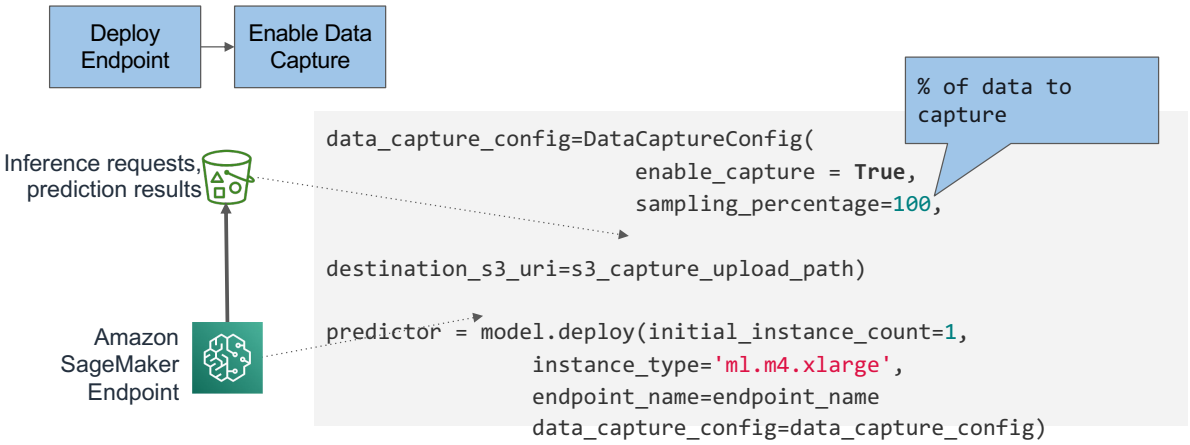
Data Quality

Monitor drift in data quality

- Monitor when inference data drifts away from baseline (training) data
- Model Monitor uses, **Deequ**, an open source library built on Apache Spark

Amazon SageMaker Model Monitor

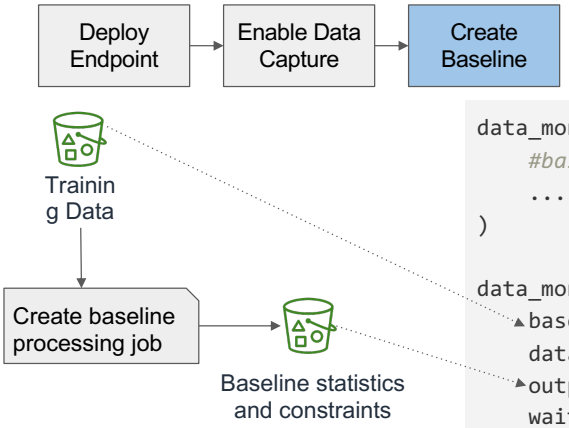
Monitor Type: Data Quality Monitor




```
data_capture_config=DataCaptureConfig(
                        enable_capture = True,
                        sampling_percentage=100,

destination_s3_uri=s3_capture_upload_path)

predictor = model.deploy(initial_instance_count=1,
                instance_type='ml.m4.xlarge',
                endpoint_name=endpoint_name
                data_capture_config=data_capture_config)
```


Amazon SageMaker Model Monitor

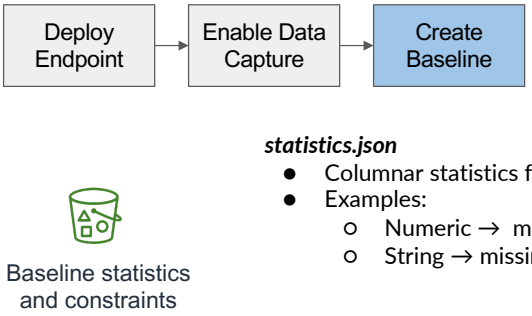
Monitor Type: Data Quality Monitor




```
data_monitor = DefaultModelMonitor(
    #baseline processing job configuration - data
    ...
)

data_monitor.suggest_baseline(
    baseline_dataset=s3train_data_uri,
    dataset_format=DatasetFormat.csv(header=True),
    output_s3_uri=baseline_results_uri,
    wait=True
)
```


Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor

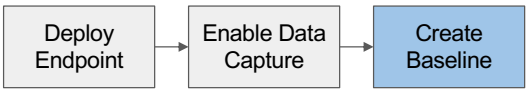


statistics.json

- Columnar statistics for each feature
- Examples:
  - Numeric → missing values, mean, min, max, distribution
  - String → missing values, distinct values, categorical distribution

Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor



Baseline statistics and constraints

statistics.json

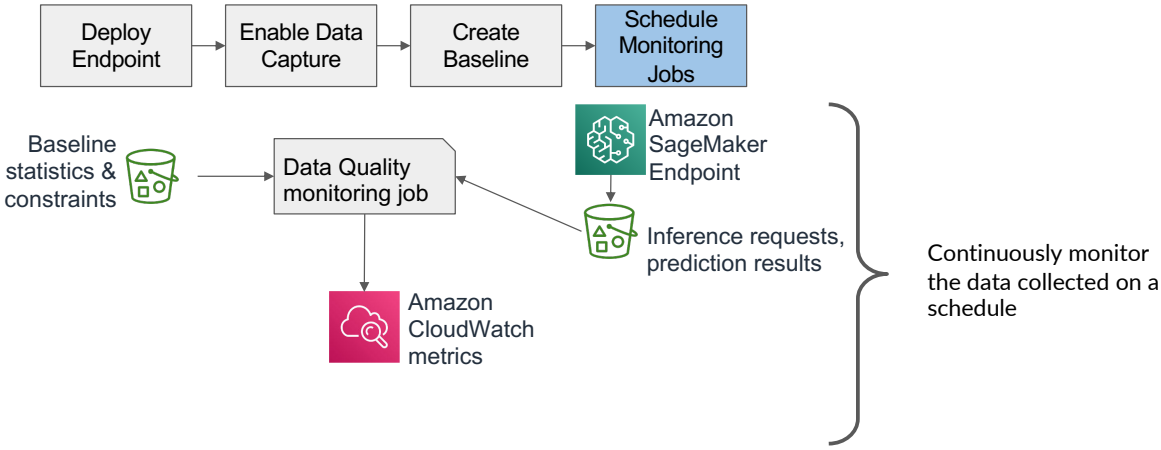
- Columnar statistics for each feature
- Examples:
  - Numeric → missing values, mean, min, max, distribution
  - String → missing values, distinct values, categorical distribution

constraints.json

- Constraints that are used to evaluate potential data drift
- Examples:
  - Numeric → non-negative
  - String → observed values

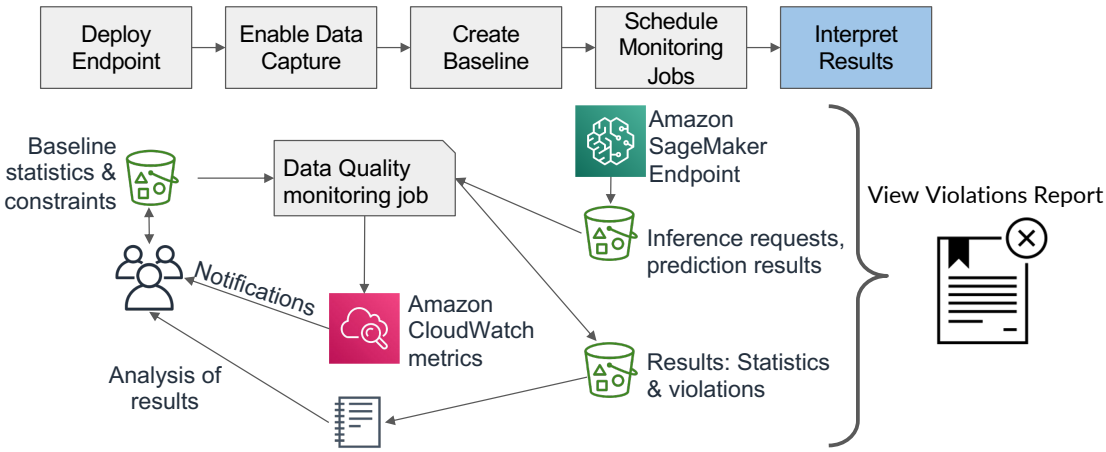
Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor



Amazon SageMaker Model Monitor

Monitor Type: Data Quality Monitor



Amazon SageMaker Model Monitor

Monitor Types

Data Drift	*Concept Drift*	*Concept Drift*	*Data Drift*
Data Quality	**Model Quality**	**Statistical Bias Drift**	**Feature Attribution Drift**
Monitor drift in data quality	Monitor drift in model quality metrics	Monitor statistical bias drift in model predictions	Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Model Quality Monitor

Concept Drift

Model Quality

Monitor drift in model quality metrics

- Monitor model quality by comparing model predictions with ground truth labels

Training Data | Model Metric *Ex. Accuracy*

VS

Ground Truth Data | Model Metric *Ex. Accuracy*

Amazon SageMaker Model Monitor

Monitor Types

Data Drift	*Concept Drift*	*Concept Drift*	*Data Drift*
Data Quality	**Model Quality**	**Statistical Bias Drift**	**Feature Attribution Drift**
Monitor drift in data quality	Monitor drift in model quality metrics	Monitor statistical bias drift in model predictions	Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Statistical Bias Drift

Concept Drift

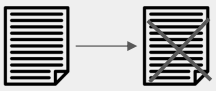
Statistical Bias Drift

Monitor statistical bias drift in model predictions

- Monitor predictions for statistical bias
- Amazon SageMaker Clarify integrates with Amazon SageMaker Model Monitor to detect statistical bias drift

Amazon SageMaker Model Monitor

Monitor Types

Data Drift	*Concept Drift*	*Concept Drift*	*Data Drift*
Data Quality	**Model Quality**	**Statistical Bias Drift**	**Feature Attribution Drift**
			
Monitor drift in data quality	Monitor drift in model quality metrics	Monitor statistical bias drift in model predictions	Monitor drift in feature attribution

Amazon SageMaker Model Monitor

Monitor Type: Feature Attribution Drift

Data Drift

Feature Attribution Drift

Monitor drift in feature attribution

- Monitor features contributing to predictions over time
- Amazon SageMaker Clarify integrates with Amazon SageMaker Model Monitor to detect feature attribution drift
- Utilizes SHAP for baselining

A/B testing, traffic shifting and autoscaling

Introduction

In this lab you will create an endpoint with multiple variants, splitting the traffic between them. Then after testing and reviewing the endpoint performance metrics, you will shift the traffic to one variant and configure it to autoscale.

Table of Contents

- 1. Create an endpoint with multiple variants
  - 1.1. Construct Docker Image URI
    - Exercise 1
  - 1.2. Create Amazon SageMaker Models
    - Exercise 2
    - Exercise 3
  - 1.3. Set up Amazon SageMaker production variants
    - Exercise 4
    - Exercise 5
  - 1.4. Configure and create endpoint
    - Exercise 6
- 2. Test model
  - 2.1. Test the model on a few sample strings
    - Exercise 7
  - 2.2. Generate traffic and review the endpoint performance metrics
- 3. Shift the traffic to one variant and review the endpoint performance metrics
  - Exercise 8
- 4. Configure one variant to autoscale

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
!conda install -q -y pytorch==1.6.0 -c pytorch
!pip install --disable-pip-version-check -q transformers==3.5.1
```

```
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... failed with initial frozen solve. Retr
ying with flexible solve.
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... done

## Package Plan ##

  environment location: /opt/conda

  added / updated specs:
    - pytorch==1.6.0


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2022.07.19 |       h06a4308_0         124 KB
    conda-4.14.0               |   py37h06a4308_0         909 KB
    cudatoolkit-10.2.89        |       hfd86e86_1       365.1 MB
    ninja-1.10.2               |       h06a4308_5           8 KB
    ninja-base-1.10.2          |       hd09550d_5         109 KB
    pytorch-1.6.0              |py3.7_cuda10.2.89_cudnn7.6.5_0       537.
7 MB  pytorch
    ------------------------------------------------------------
                                           Total:       903.9 MB

The following NEW packages will be INSTALLED:

  cudatoolkit        pkgs/main/linux-64::cudatoolkit-10.2.89-hfd86e86_1
  ninja              pkgs/main/linux-64::ninja-1.10.2-h06a4308_5
  ninja-base         pkgs/main/linux-64::ninja-base-1.10.2-hd09550d_5
  pytorch            pytorch/linux-64::pytorch-1.6.0-py3.7_cuda10.2.89_cu
dnn7.6.5_0

The following packages will be UPDATED:

  ca-certificates    conda-forge::ca-certificates-2022.6.1~ --> pkgs/main
::ca-certificates-2022.07.19-h06a4308_0

The following packages will be SUPERSEDED by a higher-priority channel:

  conda              conda-forge::conda-4.14.0-py37h89c186~ --> pkgs/main
::conda-4.14.0-py37h06a4308_0


Preparing transaction: ...working... done
Verifying transaction: ...working... done
Executing transaction: ...working... done
Retrieving notices: ...working... done
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 matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

In [4]:
```
import boto3
import sagemaker
import pandas as pd
import botocore

config = botocore.config.Config(user_agent_extra='dlai-pds/c3/w2')

# 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

cw = boto3.client(service_name='cloudwatch',
                  config=config)

autoscale = boto3.client(service_name="application-autoscaling",
                         config=config)
```

1. Create an endpoint with multiple variants

Two models trained to analyze customer feedback and classify the messages into positive (1), neutral (0), and negative (-1) sentiments are saved in the following S3 bucket paths. These `tar.gz` files contain the model artifacts, which result from model training.

In [5]:
```
model_a_s3_uri = 's3://dlai-practical-data-science/models/ab/variant_a/mo
model_b_s3_uri = 's3://dlai-practical-data-science/models/ab/variant_b/mo
```

Let's deploy an endpoint splitting the traffic between these two models 50/50 to perform A/B Testing. Instead of creating a PyTorch Model object and calling `model.deploy()` function, you will create an `Endpoint configuration` with multiple model variants. Here is the workflow you will follow to create an endpoint:

1.1. Construct Docker Image URI

You will need to create the models in Amazon SageMaker, which retrieves the URI for the pre-built SageMaker Docker image stored in Amazon Elastic Container Re gistry (ECR). Let's construct the ECR URI which you will pass into the `create_model` function later.

Set the instance type. For the purposes of this lab, you will use a relatively small instance. Please refer to this link for additional instance types that may work for your use cases outside of this lab.

In [6]:
```
inference_instance_type = 'ml.m5.large'
```

Exercise 1

Create an ECR URI using the `'PyTorch'` framework. Review other parameters of the image.

In [10]:
```
inference_image_uri = sagemaker.image_uris.retrieve(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    framework='pytorch', # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    version='1.6.0',
    instance_type=inference_instance_type,
    region=region,
    py_version='py3',
    image_scope='inference'
)
print(inference_image_uri)
```

```
763104351884.dkr.ecr.us-east-1.amazonaws.com/pytorch-inference:1.6.0-cpu-py3
```

1.2. Create Amazon SageMaker Models

Amazon SageMaker Model includes information such as the S3 location of the model, the container image that can be used for inference with that model, the execution role, and the model name.

Let's construct the model names.

In [11]:
```
import time
from pprint import pprint

timestamp = int(time.time())

model_name_a = '{}-{}'.format('a', timestamp)
model_name_b = '{}-{}'.format('b', timestamp)
```

You will use the following function to check if the model already exists in Amazon SageMaker.

In [12]:
```
def check_model_existence(model_name):
    for model in sm.list_models()['Models']:
        if model_name == model['ModelName']:
            return True
    return False
```

Exercise 2

Create an Amazon SageMaker Model based on the `model_a_s3_uri` data.

Instructions: Use `sm.create_model` function, which requires the model name, Amazon SageMaker execution role and a primary container description (`PrimaryContainer` dictionary). The `PrimaryContainer` includes the S3 bucket location of the model artifacts (`ModelDataUrl` key) and ECR URI (`Image` key).

In [13]:
```
if not check_model_existence(model_name_a):
    model_a = sm.create_model(
        ModelName=model_name_a,
        ExecutionRoleArn=role,
        PrimaryContainer={
            ### BEGIN SOLUTION - DO NOT delete this comment for grading p
            'ModelDataUrl': model_a_s3_uri, # Replace None
            'Image': inference_image_uri # Replace None
            ### END SOLUTION - DO NOT delete this comment for grading pur
        }
    )
    pprint(model_a)
else:
    print("Model {} already exists".format(model_name_a))
```

```
{'ModelArn': 'arn:aws:sagemaker:us-east-1:643403534509:model/a-1662363360
',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '74',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Mon, 05 Sep 2022 07:36:58
GMT',
                                      'x-amzn-requestid': 'ef12a4b2-eba0-
43f6-a0c5-017fdc4ee35e'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'ef12a4b2-eba0-43f6-a0c5-017fdc4ee35e'
,
                      'RetryAttempts': 0}}
```

Exercise 3

Create an Amazon SageMaker Model based on the `model_b_s3_uri` data.

Instructions: Use the example in the cell above.

In [14]:
```
if not check_model_existence(model_name_b):
    model_b = sm.create_model(
        ### BEGIN SOLUTION - DO NOT delete this comment for grading purpo
        ModelName=model_name_b, # Replace all None
        ExecutionRoleArn=role, # Replace all None
        ### END SOLUTION - DO NOT delete this comment for grading purpose
        PrimaryContainer={
            'ModelDataUrl': model_b_s3_uri,
            'Image': inference_image_uri
        }
    )
    pprint(model_b)
else:
    print("Model {} already exists".format(model_name_b))
```

```
{'ModelArn': 'arn:aws:sagemaker:us-east-1:643403534509:model/b-1662363360
',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '74',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Mon, 05 Sep 2022 07:37:40
GMT',
                                      'x-amzn-requestid': 'a43d35ae-16b2-
4393-abbd-2be881672bdd'},
                      'HTTPStatusCode': 200,
                      'RequestId': 'a43d35ae-16b2-4393-abbd-2be881672bdd'
,
                      'RetryAttempts': 0}}
```

1.3. Set up Amazon SageMaker production variants

A production variant is a packaged SageMaker Model combined with the configuration related to how that model will be hosted.

You have constructed the model in the section above. The hosting resources configuration includes information on how you want that model to be hosted: the number and type of instances, a pointer to the SageMaker package model, as well as a variant name and variant weight. A single SageMaker Endpoint can actually include multiple production variants.

Exercise 4

Create an Amazon SageMaker production variant for the SageMaker Model with the `model_name_a`.

Instructions: Use the `production_variant` function passing the `model_name_a` and instance type defined above.

```
variantA = production_variant(
    model_name=..., # SageMaker Model name
    instance_type=..., # instance type
    initial_weight=50, # traffic distribution weight
    initial_instance_count=1, # instance count
    variant_name='VariantA', # production variant name
)
```

In [15]:

```
from sagemaker.session import production_variant

variantA = production_variant(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    model_name=model_name_a, # Replace None
    instance_type=inference_instance_type, # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    initial_weight=50,
    initial_instance_count=1,
    variant_name='VariantA',
)
print(variantA)
```

```
{'ModelName': 'a-1662363360', 'InstanceType': 'ml.m5.large', 'InitialInst
anceCount': 1, 'VariantName': 'VariantA', 'InitialVariantWeight': 50}
```

Exercise 5

Create an Amazon SageMaker production variant for the SageMaker Model with the `model_name_b`.

Instructions: See the required arguments in the cell above.

In [16]:

```
variantB = production_variant(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    model_name=model_name_b, # Replace all None
    instance_type=inference_instance_type, # Replace all None
    initial_weight=50, # Replace all None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    initial_instance_count=1,
    variant_name='VariantB'
)
print(variantB)
```

```
{'ModelName': 'b-1662363360', 'InstanceType': 'ml.m5.large', 'InitialInst
anceCount': 1, 'VariantName': 'VariantB', 'InitialVariantWeight': 50}
```

1.4. Configure and create the endpoint

You will use the following functions to check if the endpoint configuration and endpoint itself already exist in Amazon SageMaker.

In [17]:
```
def check_endpoint_config_existence(endpoint_config_name):
    for endpoint_config in sm.list_endpoint_configs()['EndpointConfigs']:
        if endpoint_config_name == endpoint_config['EndpointConfigName']:
            return True
    return False

def check_endpoint_existence(endpoint_name):
    for endpoint in sm.list_endpoints()['Endpoints']:
        if endpoint_name == endpoint['EndpointName']:
            return True
    return False
```

Create the endpoint configuration by specifying the name and pointing to the two production variants that you just configured that tell SageMaker how you want to host those models.

In [18]:
```
endpoint_config_name = '{}-{}'.format('ab', timestamp)

if not check_endpoint_config_existence(endpoint_config_name):
    endpoint_config = sm.create_endpoint_config(
        EndpointConfigName=endpoint_config_name,
        ProductionVariants=[variantA, variantB]
    )
    pprint(endpoint_config)
else:
    print("Endpoint configuration {} already exists".format(endpoint_conf
```

```
{'EndpointConfigArn': 'arn:aws:sagemaker:us-east-1:643403534509:endpoint-
config/ab-1662363360',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '94',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Mon, 05 Sep 2022 07:39:22
GMT',
                                      'x-amzn-requestid': '6ae567cb-4aec-
48b0-95b5-5989cd6ad0f2'},
                      'HTTPStatusCode': 200,
                      'RequestId': '6ae567cb-4aec-48b0-95b5-5989cd6ad0f2'
,
                      'RetryAttempts': 0}}
```

Construct the endpoint name.

In [19]:
```
model_ab_endpoint_name = '{}-{}'.format('ab', timestamp)
print('Endpoint name: {}'.format(model_ab_endpoint_name))
```

```
Endpoint name: ab-1662363360
```

Exercise 6

Create an endpoint with the endpoint name and configuration defined above.

In [20]:
```
if not check_endpoint_existence(model_ab_endpoint_name):
    endpoint_response = sm.create_endpoint(
        ### BEGIN SOLUTION - DO NOT delete this comment for grading purpo
        EndpointName=model_ab_endpoint_name, # Replace None
        EndpointConfigName=endpoint_config_name # Replace None
        ### END SOLUTION - DO NOT delete this comment for grading purpose
    )
    print('Creating endpoint {}'.format(model_ab_endpoint_name))
    pprint(endpoint_response)
else:
    print("Endpoint {} already exists".format(model_ab_endpoint_name))
```

```
Creating endpoint ab-1662363360
{'EndpointArn': 'arn:aws:sagemaker:us-east-1:643403534509:endpoint/ab-166
2363360',
 'ResponseMetadata': {'HTTPHeaders': {'content-length': '81',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Mon, 05 Sep 2022 07:40:09
GMT',
                                      'x-amzn-requestid': 'b011779d-2e23-
40af-b9b0-19cf50b501cf'},
                       'HTTPStatusCode': 200,
                       'RequestId': 'b011779d-2e23-40af-b9b0-19cf50b501cf'
,
                       'RetryAttempts': 0}}
```

Review the created endpoint configuration in the AWS console.

Instructions:

- open the link
- notice that you are in the section Amazon SageMaker -> Endpoint configuration
- check the name of the endpoint configuration, its Amazon Resource Name (ARN) and production variants
- click on the production variants and check their container information: image and model data location

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

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

Review REST Endpoint configuration

Review the created endpoint in the AWS console.

Instructions:

- open the link
- notice that you are in the section Amazon SageMaker -> Endpoints
- check the name of the endpoint, its ARN and status
- below you can review the monitoring metrics such as CPU, memory and disk utilization. Further down you can see the endpoint configuration settings with its production variants

In [22]:
```
from IPython.core.display import display, HTML

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

Review SageMaker REST endpoint

Wait for the endpoint to deploy.

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

In [23]:
```
%%time

waiter = sm.get_waiter('endpoint_in_service')
waiter.wait(EndpointName=model_ab_endpoint_name)
```

```
CPU times: user 221 ms, sys: 13.6 ms, total: 234 ms
Wall time: 8min 2s
```

Wait until the ^^ endpoint ^^ is deployed

2. Test model

2.1. Test the model on a few sample strings

Here, you will pass sample strings of text to the endpoint in order to see the sentiment. You are given one example of each, however, feel free to play around and change the strings yourself!

Exercise 7

Create an Amazon SageMaker Predictor based on the deployed endpoint.

Instructions: Use the `Predictor` object with the following parameters. Please pass JSON serializer and deserializer objects here, calling them with the functions `JSONLinesSerializer()` and `JSONLinesDeserializer()`, respectively. More information about the serializers can be found here.

```
predictor = Predictor(
    endpoint_name=..., # endpoint name
    serializer=..., # a serializer object, used to encode data
for an inference endpoint
    deserializer=..., # a deserializer object, used to decode
data from an inference endpoint
    sagemaker_session=sess
)
```

In [24]:

```
from sagemaker.predictor import Predictor
from sagemaker.serializers import JSONLinesSerializer
from sagemaker.deserializers import JSONLinesDeserializer

inputs = [
    {"features": ["I love this product!"]},
    {"features": ["OK, but not great."]},
    {"features": ["This is not the right product."]},
]

predictor = Predictor(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    endpoint_name=model_ab_endpoint_name, # Replace None
    serializer=JSONLinesSerializer(), # Replace None
    deserializer=JSONLinesDeserializer(), # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
    sagemaker_session=sess
)

predicted_classes = predictor.predict(inputs)

for predicted_class in predicted_classes:
    print("Predicted class {} with probability {}".format(predicted_class
```

```
Predicted class 1 with probability 0.9605445861816406
Predicted class 0 with probability 0.5798221230506897
Predicted class -1 with probability 0.7667604684829712
```

2.2. Generate traffic and review the endpoint performance metrics

Now you will generate traffic. To analyze the endpoint performance you will review some of the metrics that Amazon SageMaker emits in CloudWatch: CPU Utilization, Latency and Invocations. Full list of namespaces and metrics can be found here. CloudWatch `get_metric_statistics` documentation can be found here.

But before that, let's create a function that will help to extract the results from CloudWatch and plot them.

In [25]:
```
def plot_endpoint_metrics_for_variants(endpoint_name,
                                       namespace_name,
                                       metric_name,
                                       variant_names,
                                       start_time,
                                       end_time):

    try:
        joint_variant_metrics = None

        for variant_name in variant_names:
            metrics = cw.get_metric_statistics( # extracts the results in
                Namespace=namespace_name, # the namespace of the metric,
                MetricName=metric_name, # the name of the metric, e.g. "C
                StartTime=start_time, # the time stamp that determines th
                EndTime=end_time, # the time stamp that determines the la
                Period=60, # the granularity, in seconds, of the returned
                Statistics=["Sum"], # the metric statistics
                Dimensions=[ # dimensions, as CloudWatch treats each uniq
                    {"Name": "EndpointName", "Value": endpoint_name},
                    {"Name": "VariantName", "Value": variant_name}
                ],
            )

            if metrics["Datapoints"]: # access the results from the disti
                df_metrics = pd.DataFrame(metrics["Datapoints"]) \
                    .sort_values("Timestamp") \
                    .set_index("Timestamp") \
                    .drop("Unit", axis=1) \
                    .rename(columns={"Sum": variant_name}) # rename the c

                if joint_variant_metrics is None:
                    joint_variant_metrics = df_metrics
                else:
                    joint_variant_metrics = joint_variant_metrics.join(df

        joint_variant_metrics.plot(title=metric_name)
    except:
        pass
```

Establish wide enough time bounds to show all the charts using the same timeframe:

In [26]:
```
from datetime import datetime, timedelta

start_time = datetime.now() - timedelta(minutes=30)
end_time = datetime.now() + timedelta(minutes=30)

print('Start Time: {}'.format(start_time))
print('End Time: {}'.format(end_time))
```

```
Start Time: 2022-09-05 07:22:22.029450
End Time: 2022-09-05 08:22:22.029644
```

Set the list of the the variant names to analyze.

In [27]:
```
variant_names = [variantA["VariantName"], variantB["VariantName"]]

print(variant_names)
```

```
['VariantA', 'VariantB']
```

Run some predictions and view the metrics for each variant.

This cell will take approximately 1-2 minutes to run.

In [28]:
```
%%time

for i in range(0, 100):
    predicted_classes = predictor.predict(inputs)
```

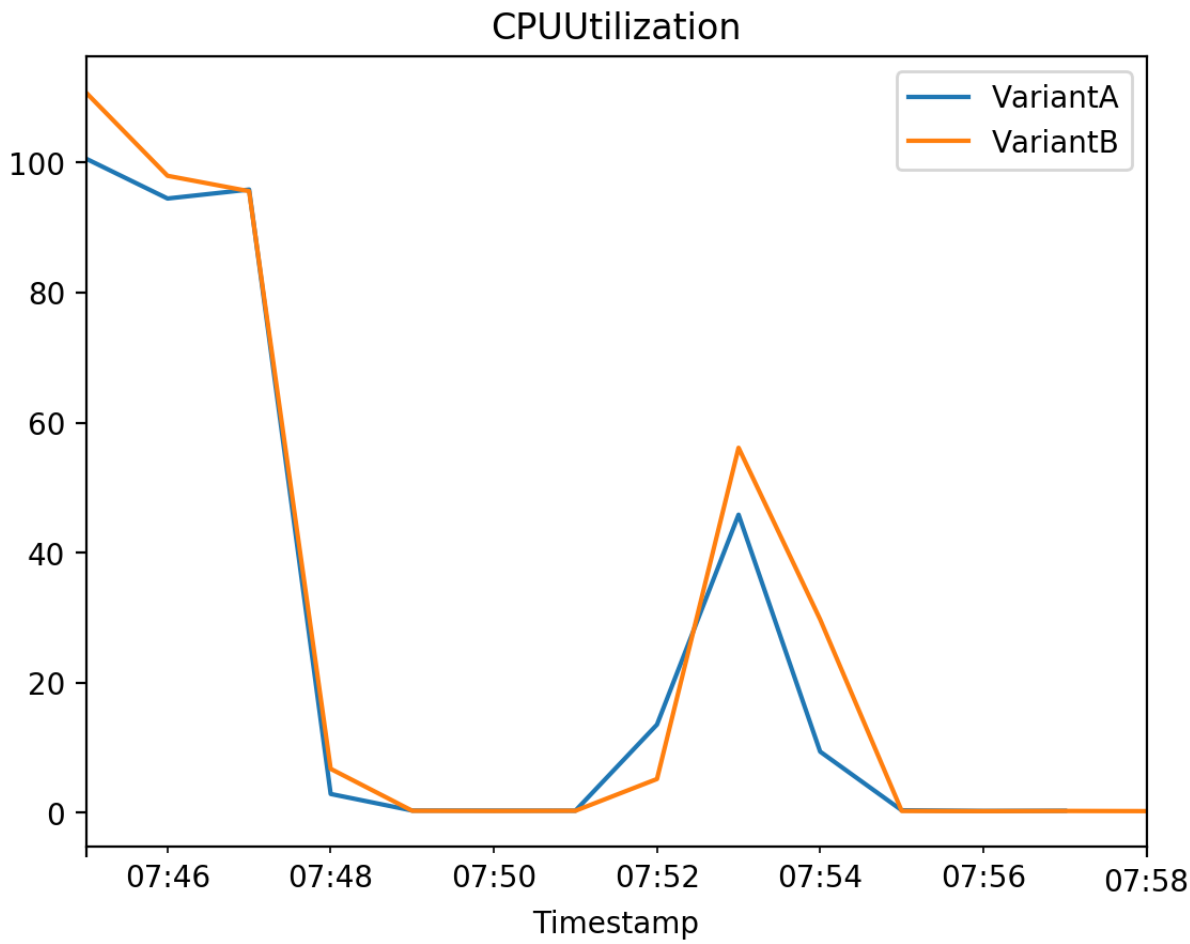
```
CPU times: user 218 ms, sys: 14.9 ms, total: 233 ms
Wall time: 1min 34s
```

Make sure the predictions ^^ above ^^ ran successfully

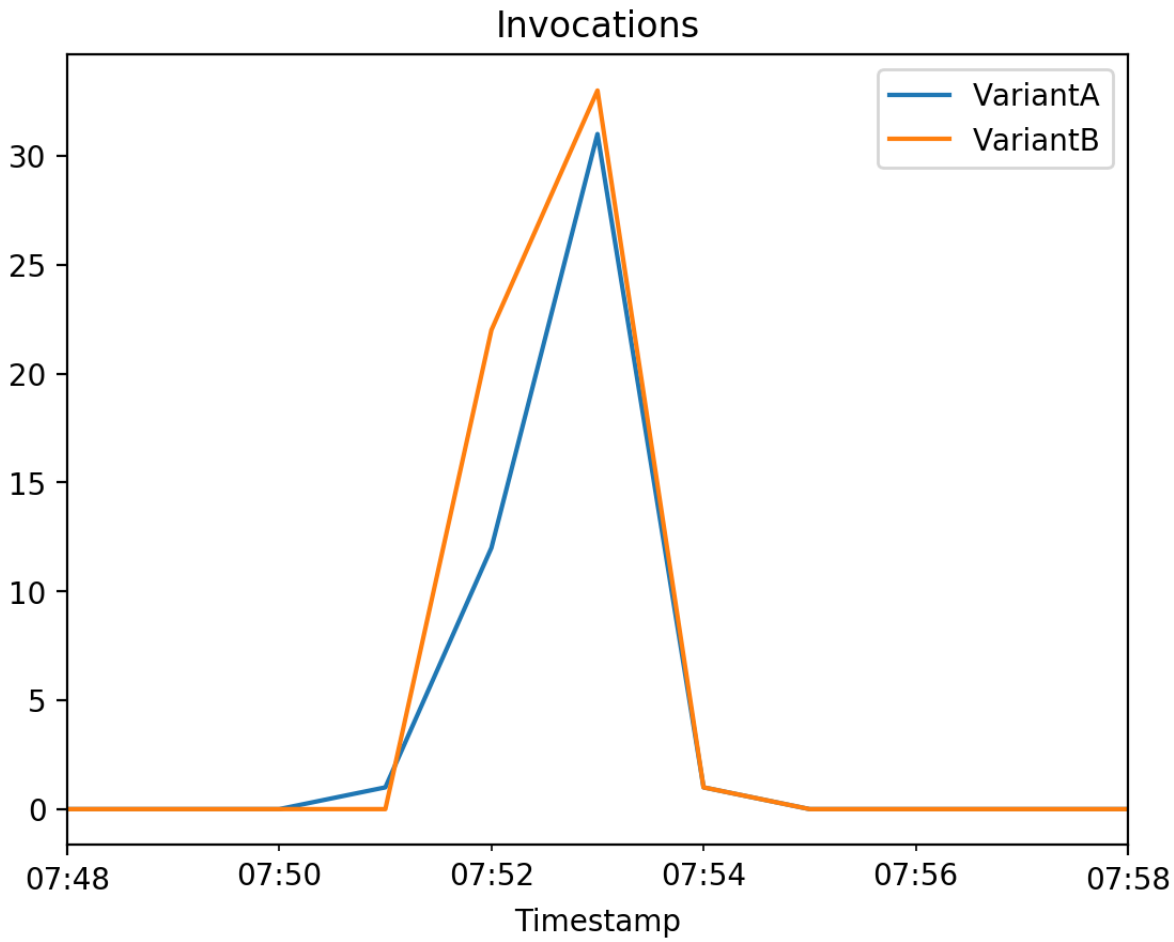
Let's query CloudWatch to get a few metrics that are split across variants. If you see `Metrics not yet available`, please be patient as metrics may take a few mins to appear in CloudWatch.

In [29]:
```
time.sleep(30) # Sleep to accomodate a slight delay in metrics gathering
```

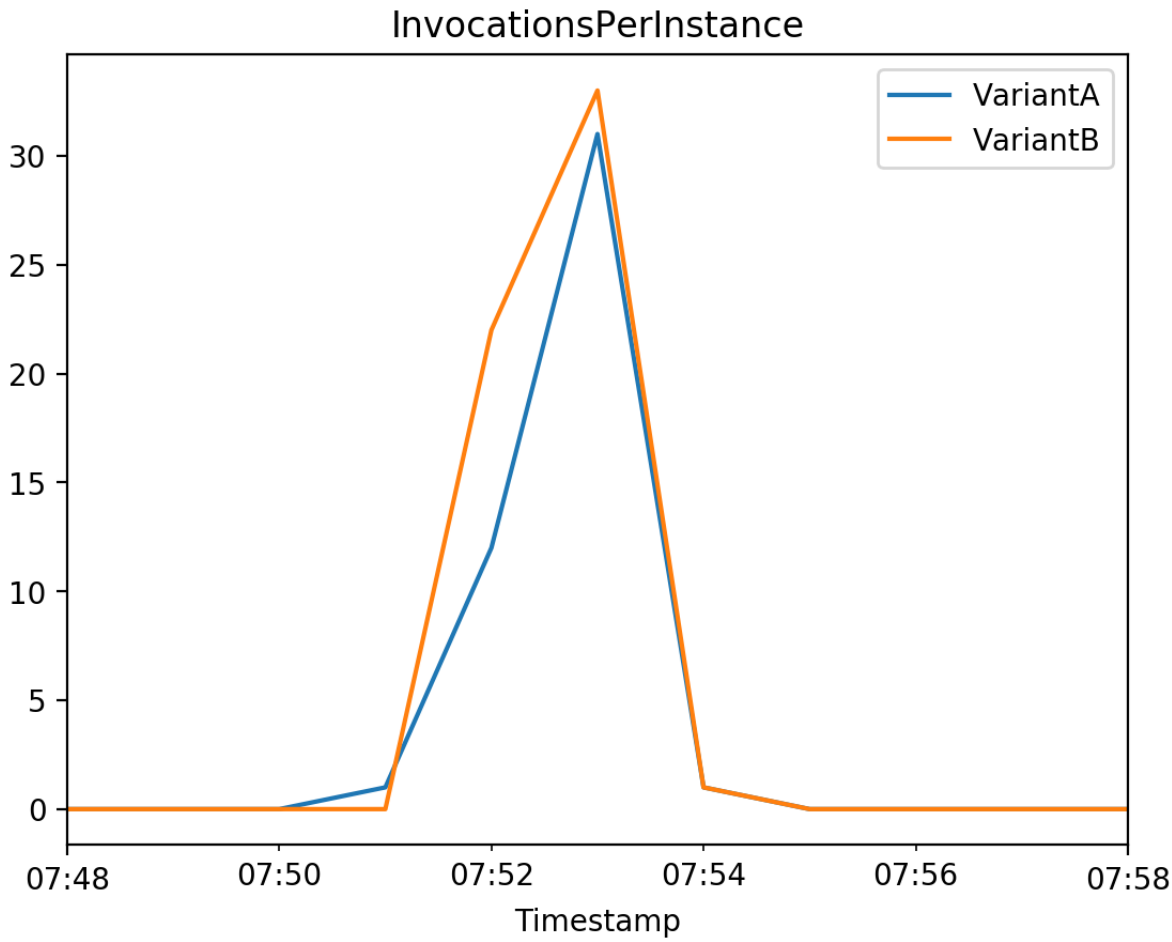
In [30]:
```
# CPUUtilization
# The sum of each individual CPU core's utilization.
# The CPU utilization of each core can range between 0 and 100. For examp
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="/aws/sagemaker/Endpoints",
    metric_name="CPUUtilization",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



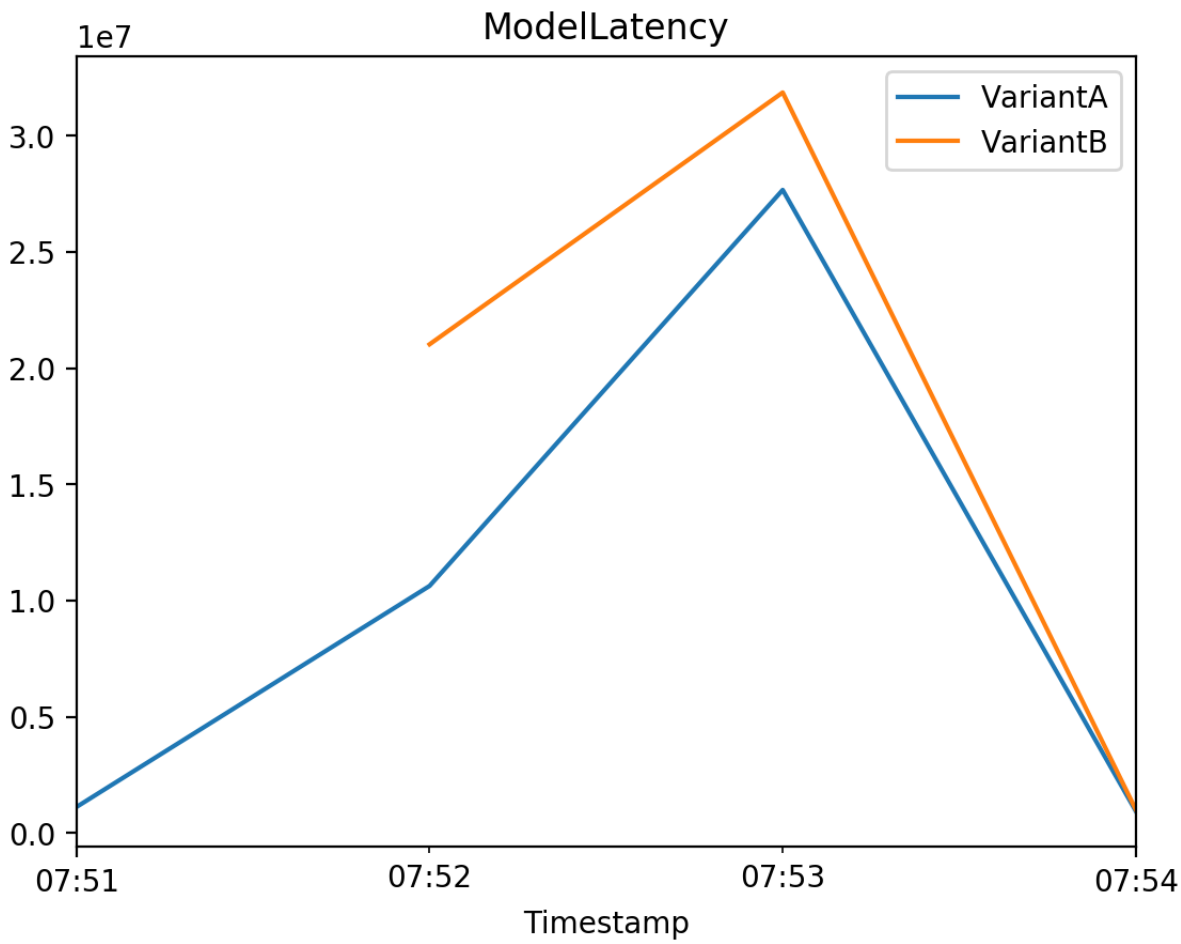
In [31]:
```
# Invocations
# The number of requests sent to a model endpoint.
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="Invocations",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



In [32]:
```
# InvocationsPerInstance
# The number of invocations sent to a model, normalized by InstanceCount
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="InvocationsPerInstance",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



In [33]:
```
# ModelLatency
# The interval of time taken by a model to respond as viewed from SageMak
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="ModelLatency",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



3. Shift the traffic to one variant and review the endpoint performance metrics

Generally, the winning model would need to be chosen. The decision would be made based on the endpoint performance metrics and some other business related evaluations. Here you can assume that the winning model is in the Variant B and shift all traffic to it.

Construct a list with the updated endpoint weights.

***No downtime** occurs during this traffic-shift activity.*

This may take a few minutes. Please be patient.

In [34]:
```
updated_endpoint_config = [
    {
        "VariantName": variantA["VariantName"],
        "DesiredWeight": 0,
    },
    {
        "VariantName": variantB["VariantName"],
        "DesiredWeight": 100,
    },
]
```

Exercise 8

Update variant weights in the configuration of the existing endpoint.

Instructions: Use the `sm.update_endpoint_weights_and_capacities` function, passing the endpoint name and list of updated weights for each of the variants that you defined above.

In [35]:
```
sm.update_endpoint_weights_and_capacities(
    ### BEGIN SOLUTION - DO NOT delete this comment for grading purposes
    EndpointName=model_ab_endpoint_name, # Replace None
    DesiredWeightsAndCapacities=updated_endpoint_config # Replace None
    ### END SOLUTION - DO NOT delete this comment for grading purposes
)
```

Out[35]:
```
{'EndpointArn': 'arn:aws:sagemaker:us-east-1:643403534509:endpoint/ab-166
2363360',
 'ResponseMetadata': {'RequestId': 'f87b7bf1-4b75-4433-8048-617932f1b48f'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'f87b7bf1-4b75-4433-8048-617932f1b4
8f',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '81',
   'date': 'Mon, 05 Sep 2022 07:59:21 GMT'},
  'RetryAttempts': 0}}
```

Wait for the ^^ endpoint update ^^ to complete above

This may take a few minutes. Please be patient.

*There is **no downtime** while the update is applying.*

While waiting for the update (or afterwards) you can review the endpoint in the AWS console.

Instructions:

- open the link
- notice that you are in the section Amazon SageMaker -> Endpoints
- check the name of the endpoint, its ARN and status (`Updating` or `InService`)
- below you can see the endpoint runtime settings with the updated weights

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

In [37]:
```
waiter = sm.get_waiter("endpoint_in_service")
waiter.wait(EndpointName=model_ab_endpoint_name)
```

Run some more predictions and view the metrics for each variant.

This cell will take approximately 1-2 minutes to run.

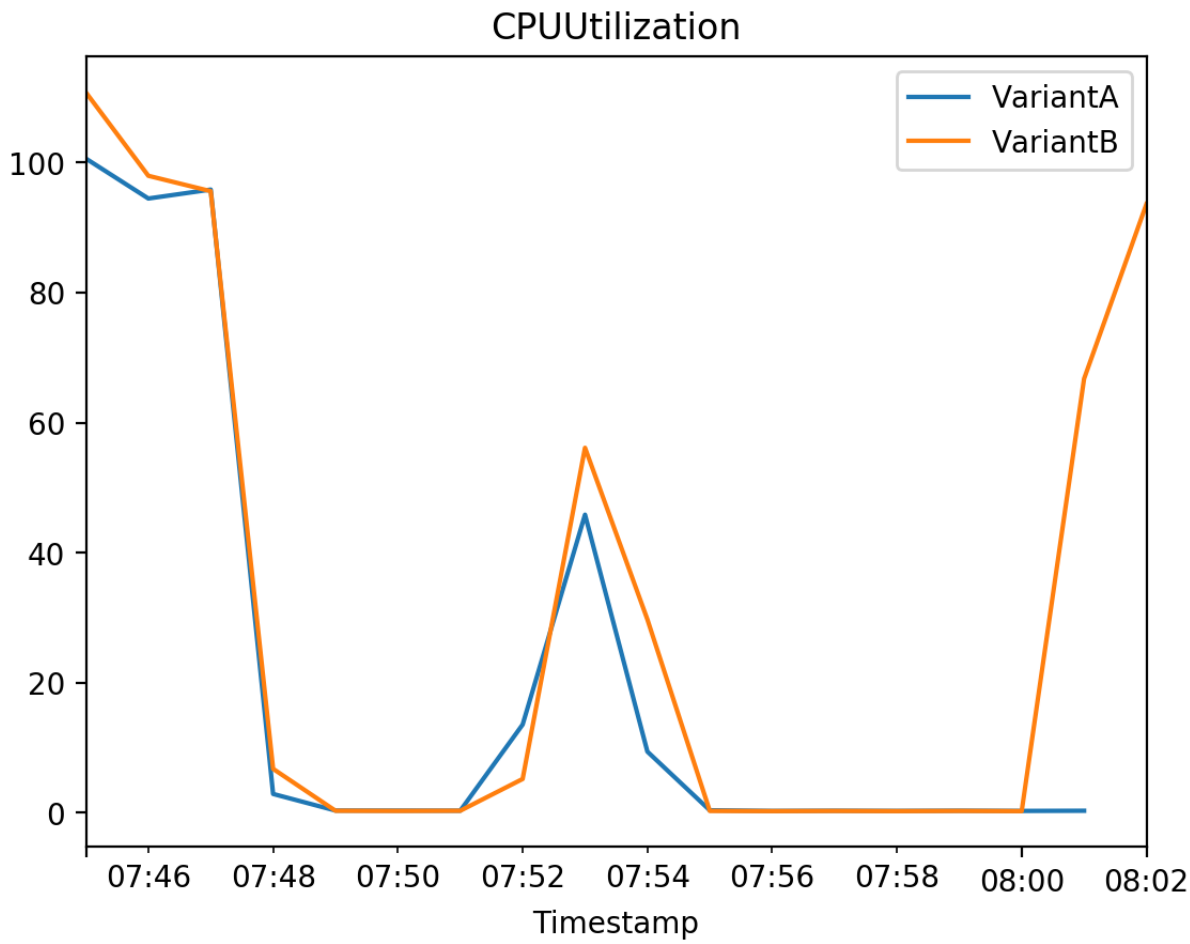
In [38]:
```
%%time

for i in range(0, 100):
    predicted_classes = predictor.predict(inputs)
```

```
CPU times: user 237 ms, sys: 14 ms, total: 252 ms
Wall time: 1min 36s
```

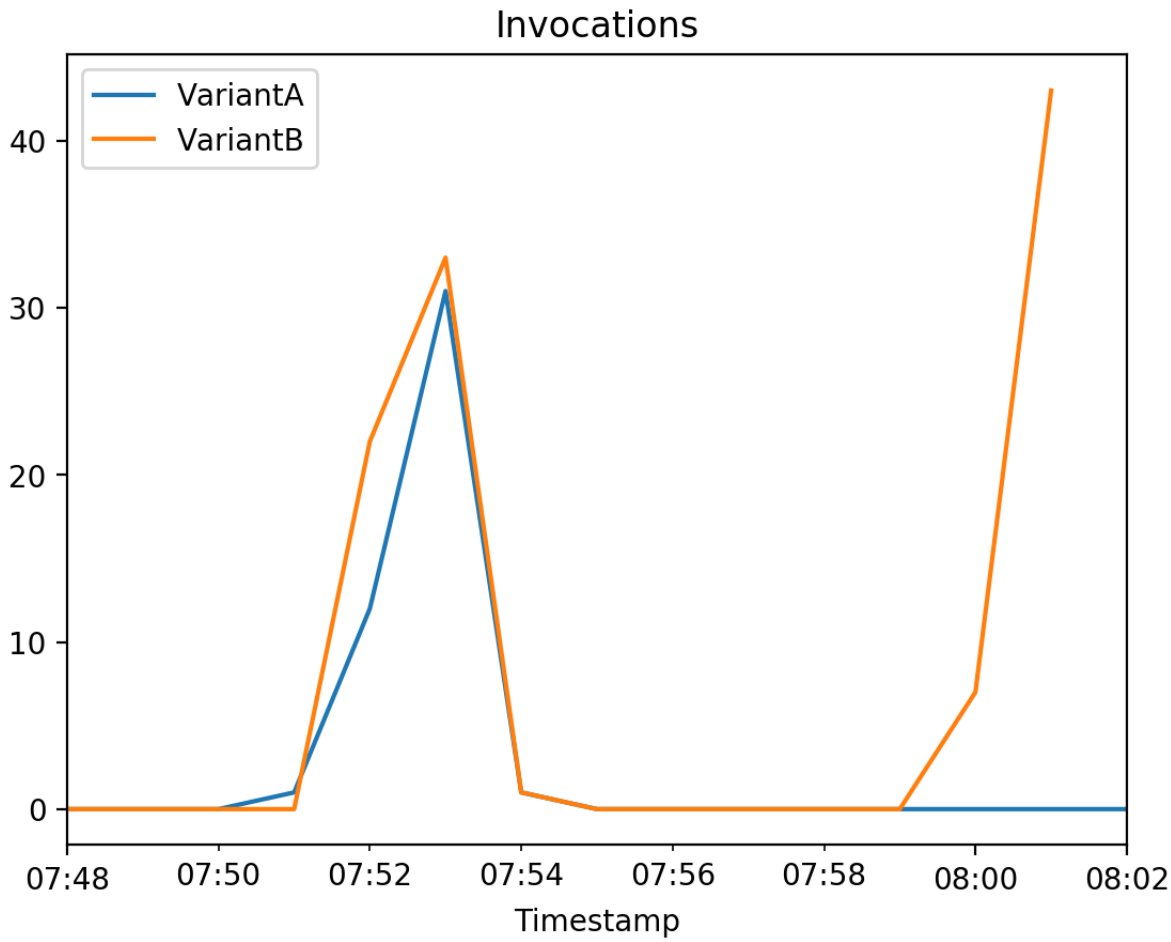
Make sure the predictions ^^ above ^^ ran successfully

If you see `Metrics not yet available`, please be patient as metrics may take a few minutes to appear in CloudWatch. Compare the results with the plots above.

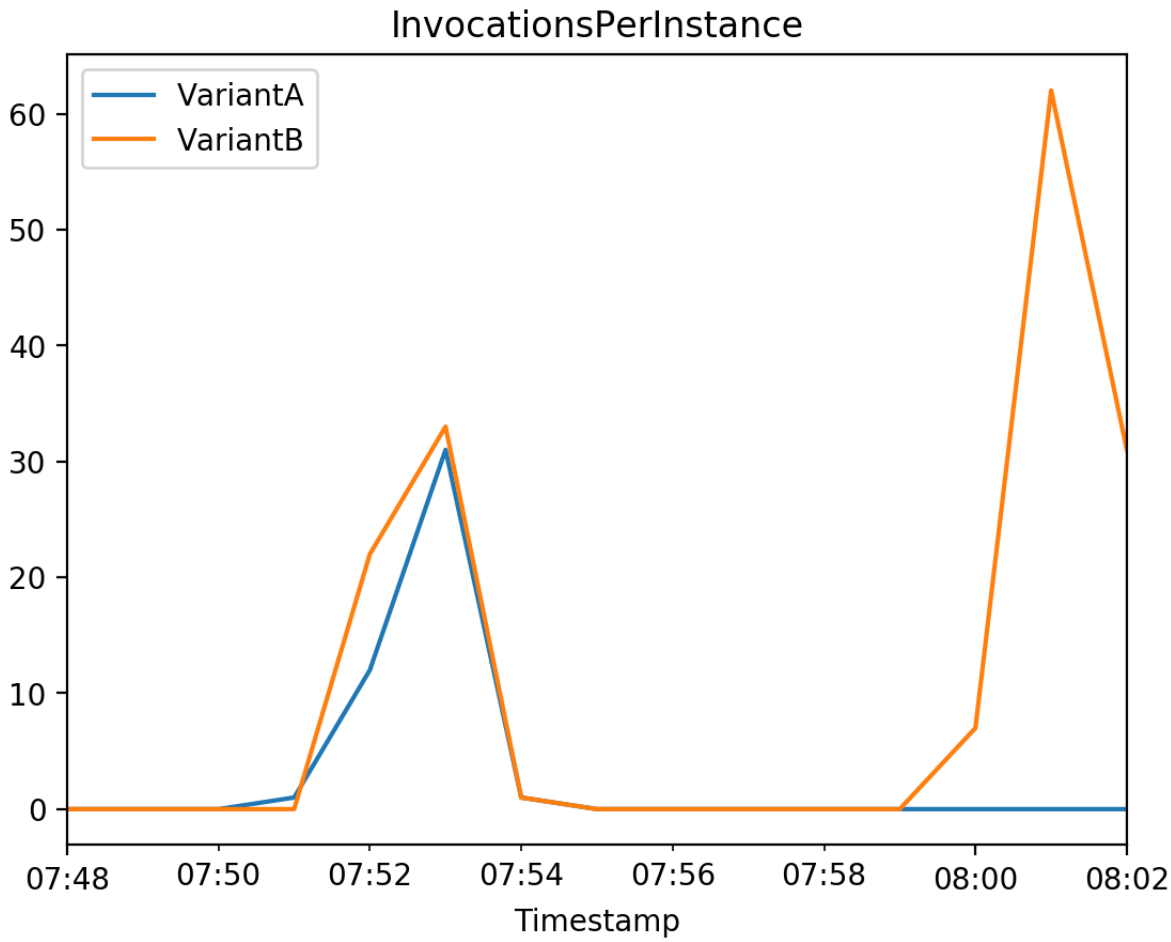
In [39]:
```
# CPUUtilization
# The sum of each individual CPU core's utilization.
# The CPU utilization of each core can range between 0 and 100. For examp
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="/aws/sagemaker/Endpoints",
    metric_name="CPUUtilization",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



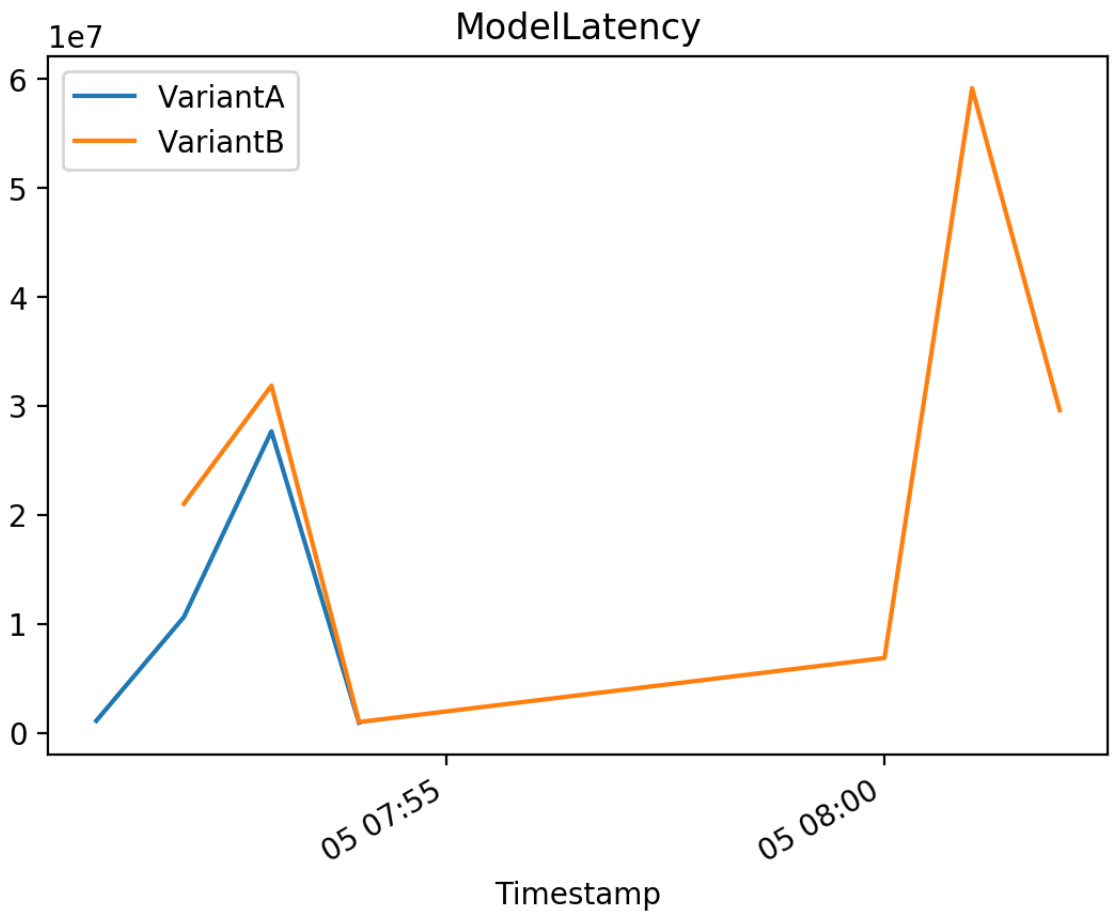
In [40]:
```
# Invocations
# The number of requests sent to a model endpoint.
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="Invocations",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



In [41]:
```
# InvocationsPerInstance
# The number of invocations sent to a model, normalized by InstanceCount
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="InvocationsPerInstance",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



In [42]:
```
# ModelLatency
# The interval of time taken by a model to respond as viewed from SageMak
plot_endpoint_metrics_for_variants(
    endpoint_name=model_ab_endpoint_name,
    namespace_name="AWS/SageMaker",
    metric_name="ModelLatency",
    variant_names=variant_names,
    start_time=start_time,
    end_time=end_time
)
```



4. Configure one variant to autoscale

Let's configure Variant B to autoscale. You would not autoscale Variant A since no traffic is being passed to it at this time.

First, you need to define a scalable target. It is an AWS resource and in this case you want to scale a `sagemaker` resource as indicated in the `ServiceNameSpace` parameter. Then the `ResourceId` is a SageMaker Endpoint. Because autoscaling is used by other AWS resources, you'll see a few parameters that will remain static for scaling SageMaker Endpoints. Thus the `ScalableDimension` is a set value for SageMaker Endpoint scaling.

You also need to specify a few key parameters that control the min and max behavior for your Machine Learning instances. The `MinCapacity` indicates the minimum number of instances you plan to scale in to. The `MaxCapacity` is the maximum number of instances you want to scale out to. So in this case you always want to have at least 1 instance running and a maximum of 2 during peak periods.

In [43]:
```
autoscale.register_scalable_target(
    ServiceNamespace="sagemaker",
    ResourceId="endpoint/" + model_ab_endpoint_name + "/variant/VariantB"
    ScalableDimension="sagemaker:variant:DesiredInstanceCount",
    MinCapacity=1,
    MaxCapacity=2,
    RoleARN=role,
    SuspendedState={
        "DynamicScalingInSuspended": False,
        "DynamicScalingOutSuspended": False,
        "ScheduledScalingSuspended": False,
    },
)
```

Out[43]:
```
{'ResponseMetadata': {'RequestId': 'ce2fa889-5861-43f5-8704-95a5e112dc5e'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'ce2fa889-5861-43f5-8704-95a5e112dc
5e',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '2',
   'date': 'Mon, 05 Sep 2022 08:02:57 GMT'},
  'RetryAttempts': 0}}
```

In [44]:
```
waiter = sm.get_waiter("endpoint_in_service")
waiter.wait(EndpointName=model_ab_endpoint_name)
```

Check that the parameters from the function above are in the description of the scalable target:

In [45]:
```
autoscale.describe_scalable_targets(
    ServiceNamespace="sagemaker",
    MaxResults=100,
)
```

Out[45]:
```
{'ScalableTargets': [{'ServiceNamespace': 'sagemaker',
   'ResourceId': 'endpoint/ab-1662363360/variant/VariantB',
   'ScalableDimension': 'sagemaker:variant:DesiredInstanceCount',
   'MinCapacity': 1,
   'MaxCapacity': 2,
   'RoleARN': 'arn:aws:iam::643403534509:role/aws-service-role/sagemaker.
application-autoscaling.amazonaws.com/AWSServiceRoleForApplicationAutoSca
ling_SageMakerEndpoint',
   'CreationTime': datetime.datetime(2022, 9, 5, 8, 2, 57, 340000, tzinfo
=tzlocal()),
   'SuspendedState': {'DynamicScalingInSuspended': False,
    'DynamicScalingOutSuspended': False,
    'ScheduledScalingSuspended': False}}],
 'ResponseMetadata': {'RequestId': '18931257-8e70-408d-8b37-2f0a37af4dd9'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '18931257-8e70-408d-8b37-2f0a37af4d
d9',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '521',
   'date': 'Mon, 05 Sep 2022 08:03:04 GMT'},
  'RetryAttempts': 0}}
```

Define and apply scaling policy using the `put_scaling_policy` function. The scaling policy provides additional information about the scaling behavior for your instance. `TargetTrackingScaling` refers to a specific autoscaling type supported by SageMaker, that uses a scaling metric and a target value as the indicator to scale.

In the scaling policy configuration, you have the predefined metric `PredefinedMetricSpecification` which is the number of invocations on your instance and the `TargetValue` which indicates the number of invocations per ML instance you want to allow before triggering your scaling policy. A scale out cooldown of 60 seconds means that after autoscaling successfully scales out it starts to calculate the cooldown time. The scaling policy won't increase the desired capacity again until the cooldown period ends.

The scale in cooldown setting of 300 seconds means that SageMaker will not attempt to start another cooldown policy within 300 seconds of when the last one completed.

In [46]:
```
autoscale.put_scaling_policy(
    PolicyName="bert-reviews-autoscale-policy",
    ServiceNamespace="sagemaker",
    ResourceId="endpoint/" + model_ab_endpoint_name + "/variant/VariantB"
    ScalableDimension="sagemaker:variant:DesiredInstanceCount",
    PolicyType="TargetTrackingScaling",
    TargetTrackingScalingPolicyConfiguration={
        "TargetValue": 2.0, # the number of invocations per ML instance y
        "PredefinedMetricSpecification": {
            "PredefinedMetricType": "SageMakerVariantInvocationsPerInstan
        },
        "ScaleOutCooldown": 60, # wait time, in seconds, before beginning
        "ScaleInCooldown": 300, # wait time, in seconds, before beginning
    },
)
```

Out[46]:
```
{'PolicyARN': 'arn:aws:autoscaling:us-east-1:643403534509:scalingPolicy:f
1674af7-74ec-45a8-8a77-e5c3b60e7dfe:resource/sagemaker/endpoint/ab-166236
3360/variant/VariantB:policyName/bert-reviews-autoscale-policy',
 'Alarms': [{'AlarmName': 'TargetTracking-endpoint/ab-1662363360/variant/
VariantB-AlarmHigh-35bf164b-d15a-4d65-b6af-7e66aadaa965',
   'AlarmARN': 'arn:aws:cloudwatch:us-east-1:643403534509:alarm:TargetTra
cking-endpoint/ab-1662363360/variant/VariantB-AlarmHigh-35bf164b-d15a-4d6
5-b6af-7e66aadaa965'},
  {'AlarmName': 'TargetTracking-endpoint/ab-1662363360/variant/VariantB-A
larmLow-9b440097-6241-4337-a5f2-95cf0fb9e5d8',
   'AlarmARN': 'arn:aws:cloudwatch:us-east-1:643403534509:alarm:TargetTra
cking-endpoint/ab-1662363360/variant/VariantB-AlarmLow-9b440097-6241-4337
-a5f2-95cf0fb9e5d8'}],
 'ResponseMetadata': {'RequestId': '4039eedd-3e05-4f20-a98e-d4352ecb190b'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': '4039eedd-3e05-4f20-a98e-d4352ecb19
0b',
   'content-type': 'application/x-amz-json-1.1',
   'content-length': '780',
   'date': 'Mon, 05 Sep 2022 08:03:10 GMT'},
  'RetryAttempts': 0}}
```

In [47]:
```
waiter = sm.get_waiter("endpoint_in_service")
waiter.wait(EndpointName=model_ab_endpoint_name)
```

Generate traffic again and review the endpoint in the AWS console.

This cell will take approximately 1-2 minutes to run.

In [48]:
```
%%time

for i in range(0, 100):
    predicted_classes = predictor.predict(inputs)
```

```
CPU times: user 238 ms, sys: 8.39 ms, total: 246 ms
Wall time: 1min 36s
```

Review the autoscaling:

- open the link
- notice that you are in the section Amazon SageMaker -> Endpoints
- below you can see the endpoint runtime settings with the instance counts. You can run the predictions multiple times to observe the increase of the instance count to 2

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

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

Upload the notebook into S3 bucket for grading purposes.

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

In [ ]:
```
!aws s3 cp ./C3_W2_Assignment.ipynb s3://$bucket/C3_W2_Assignment_Learner
```

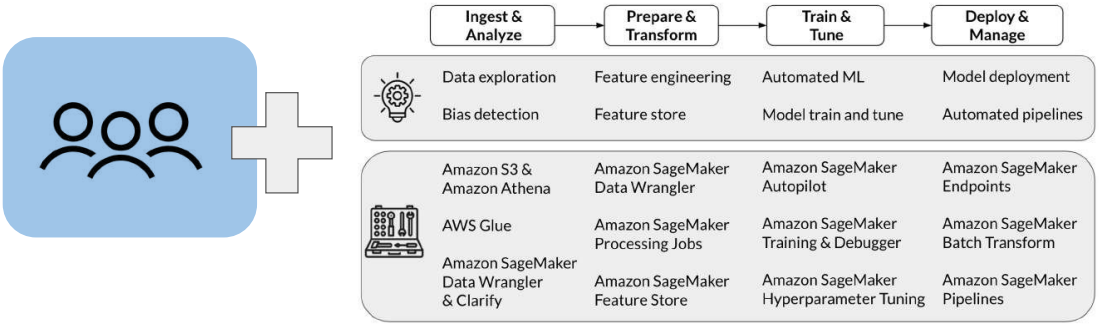
In [ ]:

DeepLearning.AI

aws

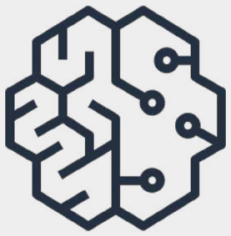
Data Labeling and Human-in-the-Loop Pipelines

Human & AI Collaboration

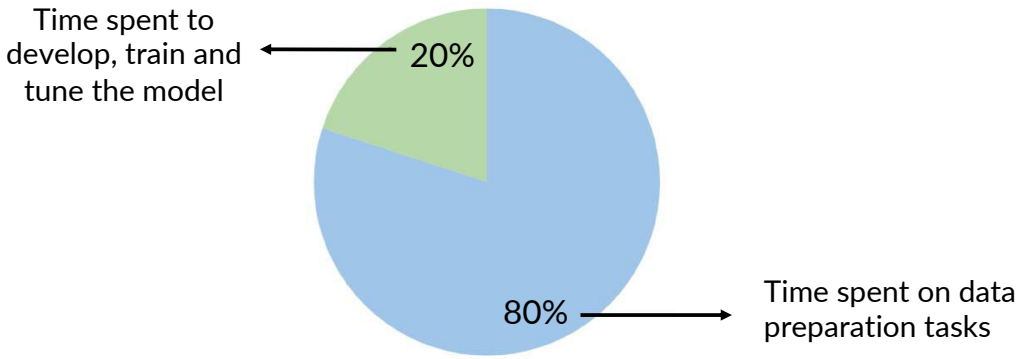


Human & AI Collaboration

Data Labeling



Preparing Data Takes Time



What Is Data Labeling?

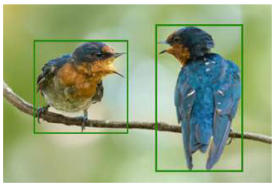
- Process of identifying raw data and adding one or more meaningful and informative labels
- Labels provide context for machine learning models to learn from (supervised learning)
- Correctly labeled datasets are often called "ground truth"

Common Types Of Data Labeling

DeepLearning.AI aws

Image Data

Single-Label Classification


Bounding Box

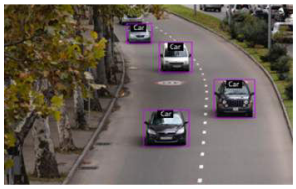
Multi-Label Classification

Semantic Segmentation

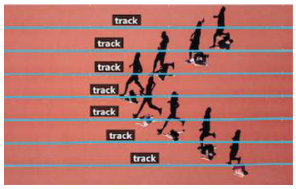
DeepLearning.AI aws

Video Data

Classification


Bounding box

Polygon

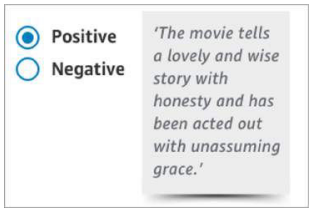


Polyline

Key point

DeepLearning.AI aws

Text Data



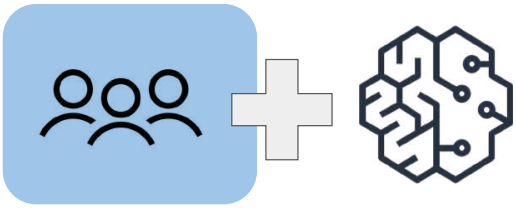
Single-Label Classification

Multi-Label Classification

Named Entity Recognition

Challenges

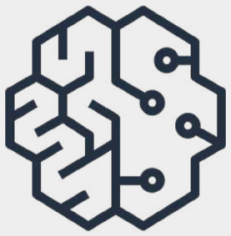
How Can Data Labeling Be Done Efficiently?



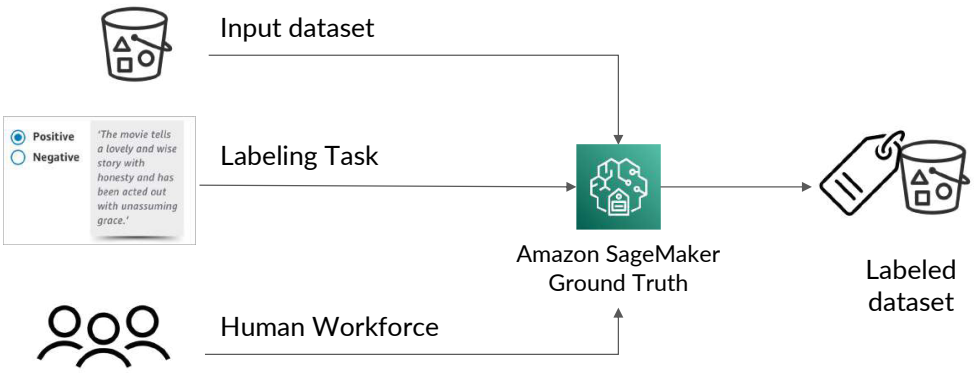
- Access to additional human workforces
- Automated data labeling capabilities
- Assistive labeling features

Data Labeling

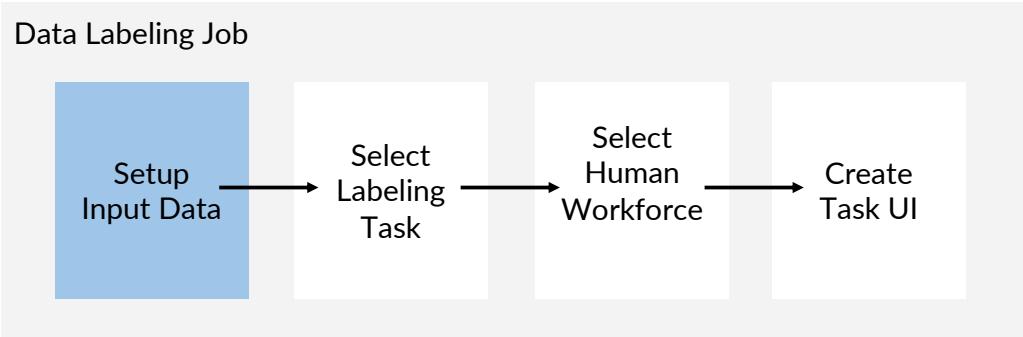
with Amazon SageMaker Ground Truth



Amazon SageMaker Ground Truth



Steps



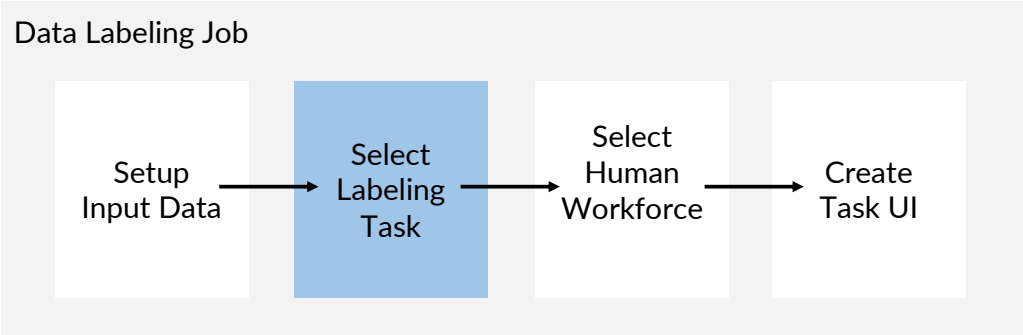
Setup Input Data

- Automated data setup
  - Provide the S3 location of the dataset you want labeled
- Manual data setup
  - Provide the S3 location of a file (an input manifest file) that identifies the data objects you want labeled

```
{"source-ref": "S3 bucket location 1"}
   ...
{"source-ref": "S3 bucket location n"}
```

```
{"source": "I love this product"}
   ...
{"source": "It is ok"}
```


Steps



Select Labeling Task

- Image Data
- Video Data
- Text Data
- Custom

Image Classification (Single Label)
Image Classification (Multi-Label)
Bounding Box
Semantic Segmentation
Label Verification

Select Labeling Task

- Image Data
- Video Data
- Text Data
- Custom

Video Clip Classification

Video Object Detection/Tracking

Bounding Box
Polygon
Polyline
Key Point

Select Labeling Task

- Image Data
- Video Data
- Text Data
- Custom

Text Classification (Single Label)

Text Classification (Multi-Label)

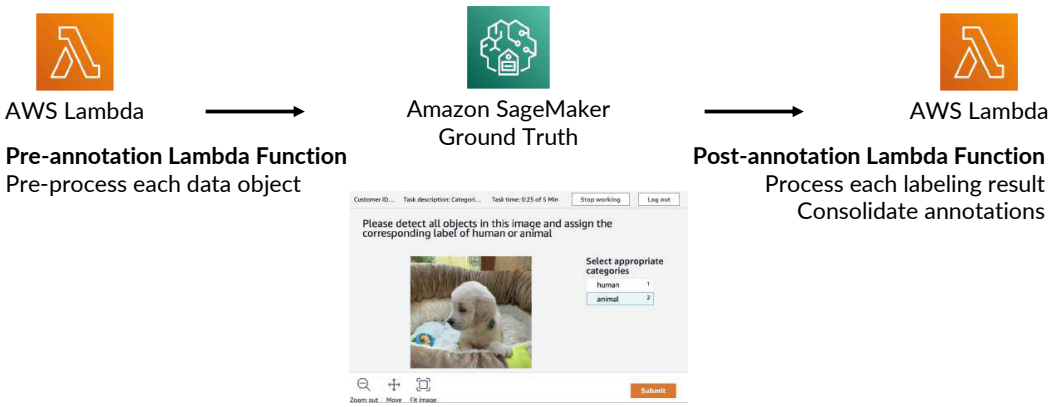
Named Entity Recognition

Select Labeling Task

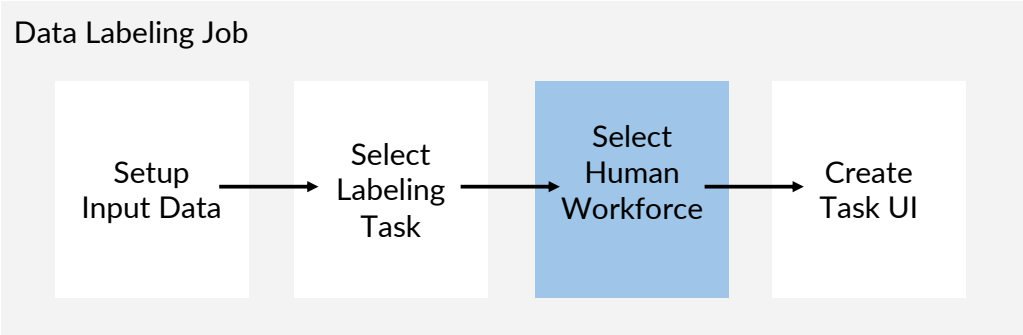
- Image Data
- Video Data
- Text Data
- Custom

Your custom labeling task

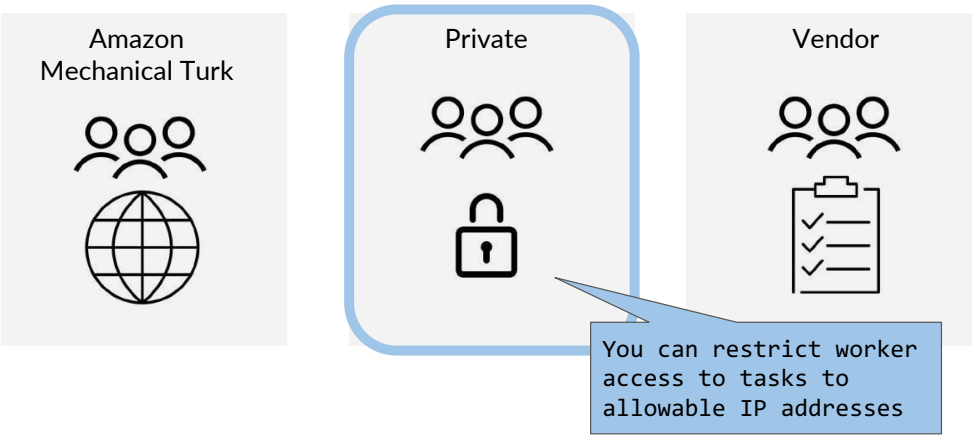
Custom Labeling Task



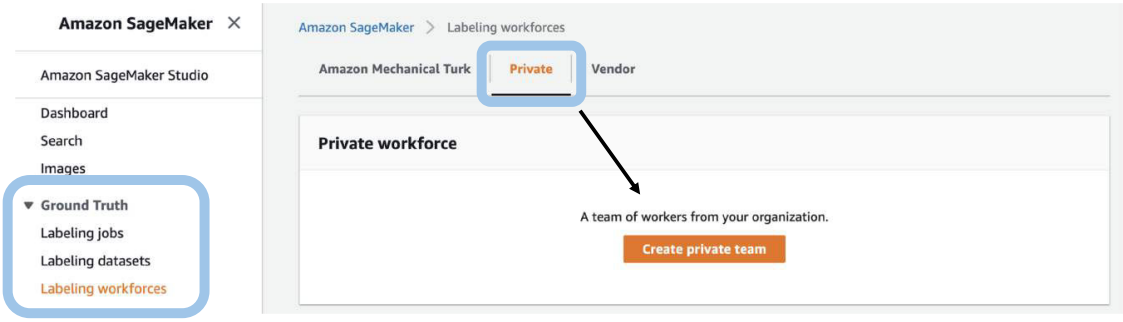
Steps



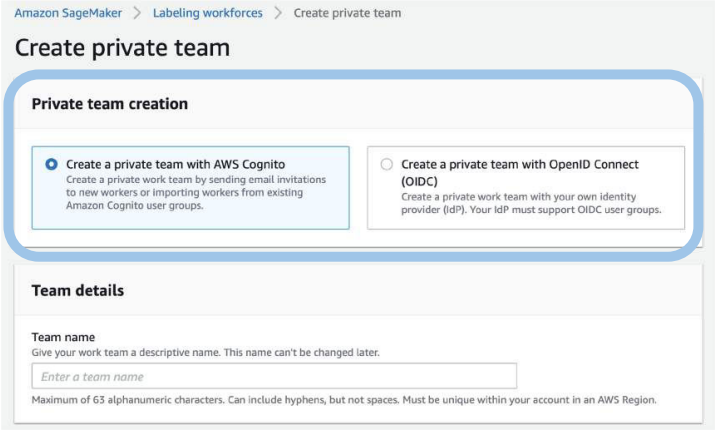
Human Workforce Options



Setup Private Workforce (UI)

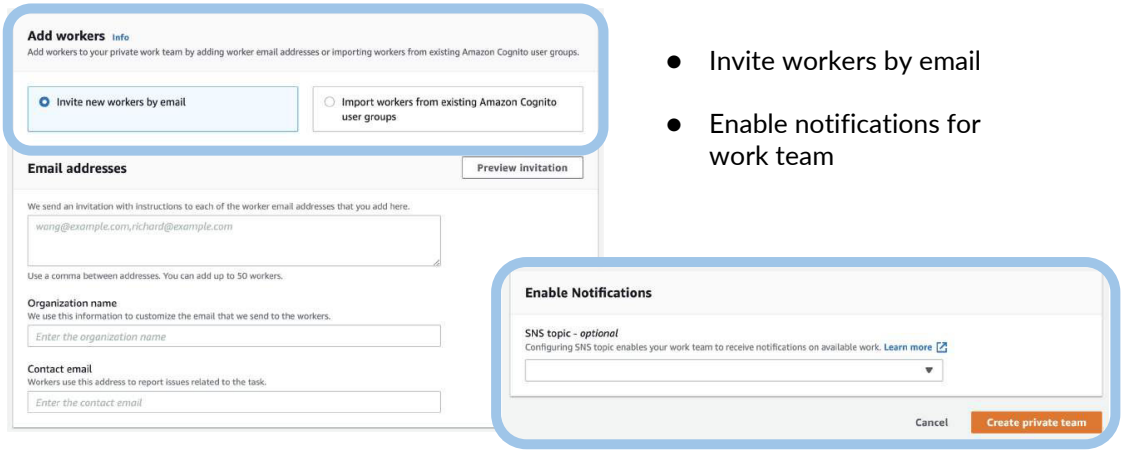


Setup Private Workforce (UI)



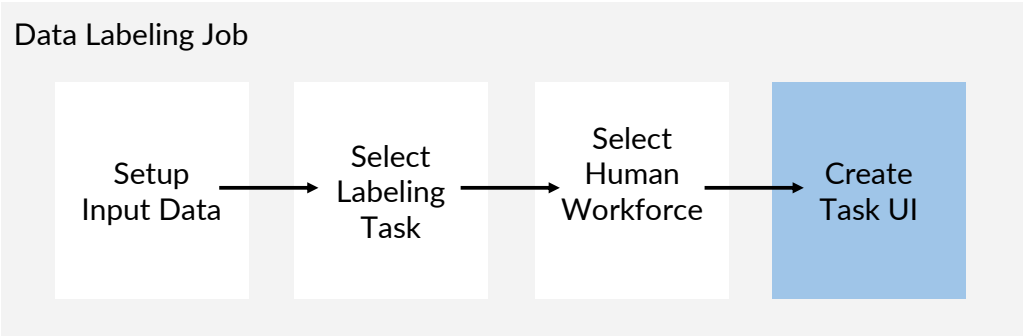
- User authentication via Amazon Cognito or OpenID Connect (OIDC)
- You can connect Amazon Cognito to your enterprise identity provider

Setup Private Workforce (UI)



- Invite workers by email
- Enable notifications for work team

Steps



Human Task UI

- Combination of HTML, CSS, JavaScript, the Liquid templating language, and Crowd HTML Elements.
- Use built-in templates, or define custom task UI

```
<script src="https://assets.crowd.aws/crowd-html-elements.js"></script>

<crowd-form>
          ...
</crowd-form>
```


Sample Task UI For Text Classification

```
<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 (negative),
                                                                 0 (neutral), and
1 (positive)">

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

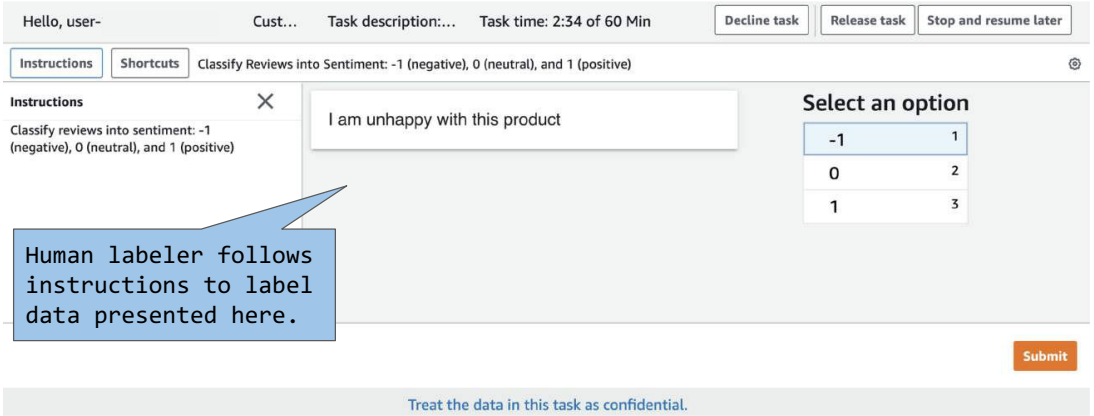
Customize instructions to match product reviews text classification

Sample Task UI For Text Classification

```
    ...
        <full-instructions header="Classify reviews into sentiment:
                                                                -1 (negative), 0
(neutral), and 1 (positive)">
            <p><strong>1</strong>: joy, excitement, delight</p>
            <p><strong>0</strong>: neither positive or negative</p>
            <p><strong>-1</strong>: anger, sarcasm, anxiety</p>
        </full-instructions>
        <short-instructions>
            Classify reviews into sentiment:
                      -1 (negative), 0 (neutral), and 1 (positive)
        </short-instructions>
    </crowd-classifier></crowd-form>
```

Provide more detailed instructions.

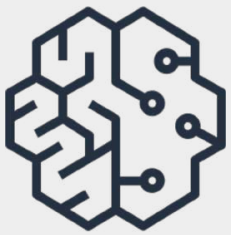
Define Task UI For Text Classification



Human labeler follows instructions to label data presented here.

Data Labeling

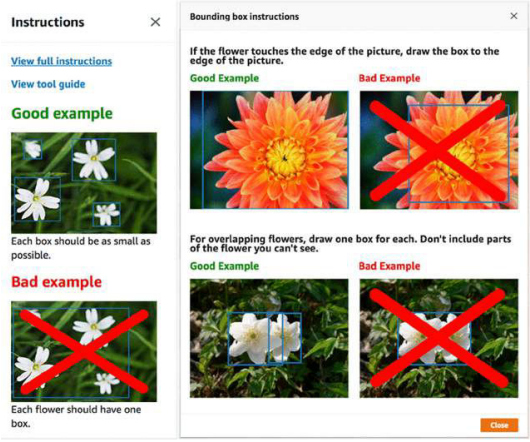
Best Practices



Best Practices

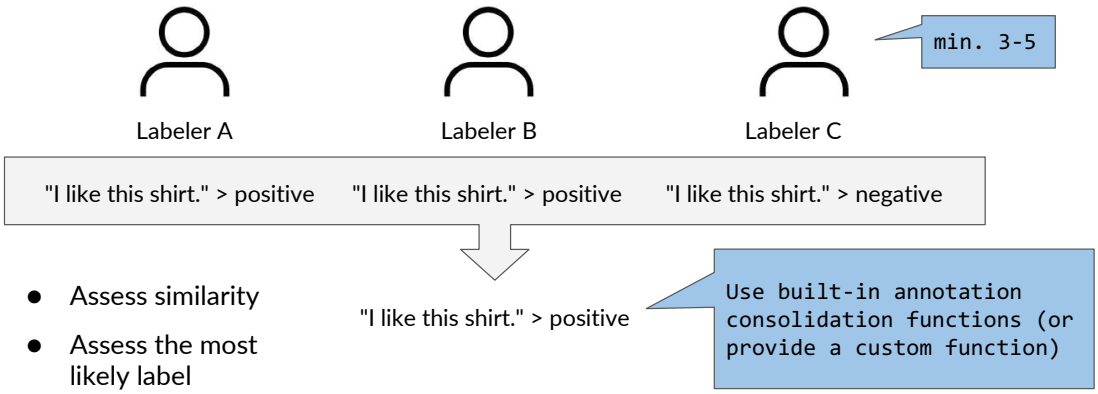
- Provide clear instructions
- Consolidate annotations to improve label quality
- Verify and adjust labels
- Use automated data labeling on large datasets (Active Learning)
- Re-use prior labeling jobs to create hierarchical labels

Provide Clear Instructions



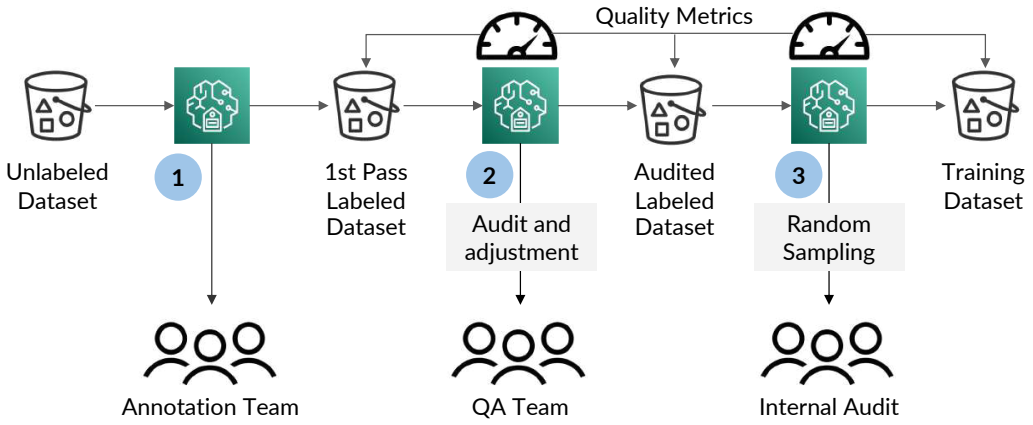
- Provide examples of good and bad annotations
- Only show relevant labels
- Use multiple workers per task

Consolidate Annotations To Improve Label Quality

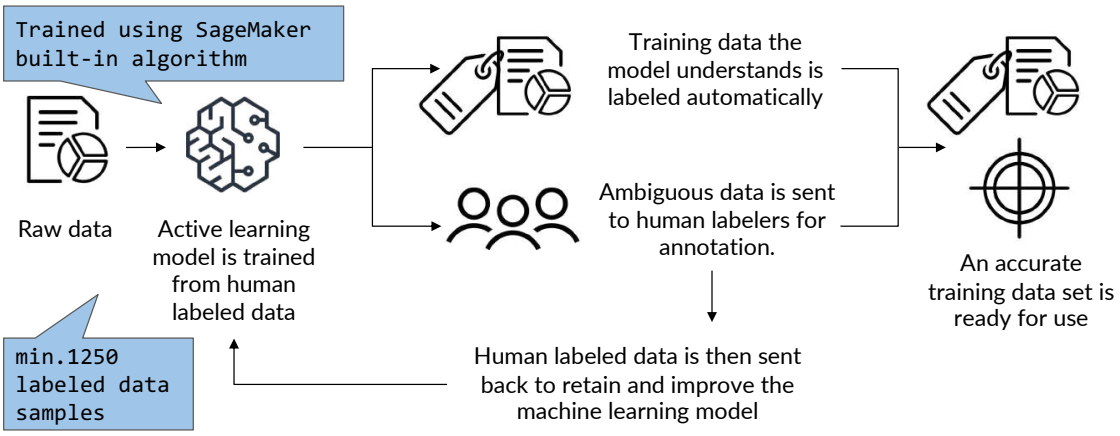


- Assess similarity
- Assess the most likely label

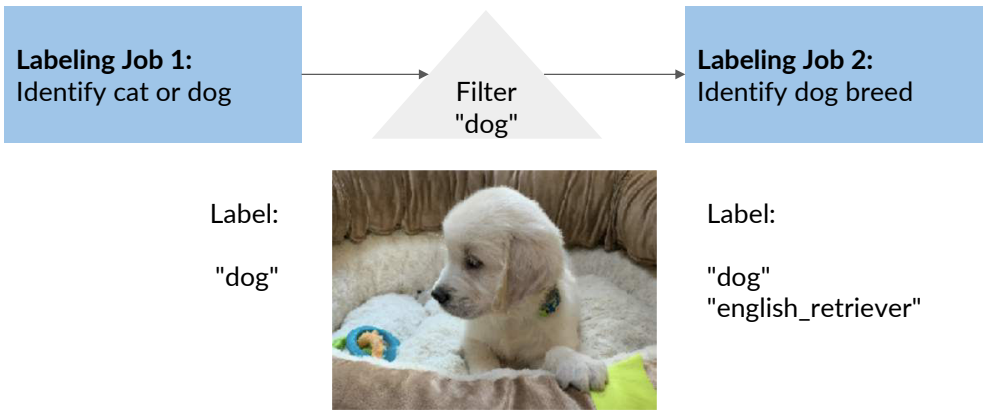
Verify And Adjust Labels



Use Automated Data Labeling On Large Datasets



Re-Use Prior Labeling Jobs To Create Hierarchical Labels



Human-In-The-Loop Pipelines

Image Classification

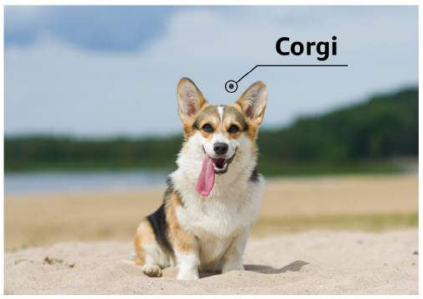
Task: Verify the appropriate dog breed has been classified in the image

1. Read the task carefully and inspect the image.
2. Read the options and review the examples to understand more about the dog breed categories.
3. Choose the appropriate dog breed category that best suits the image.

Examples

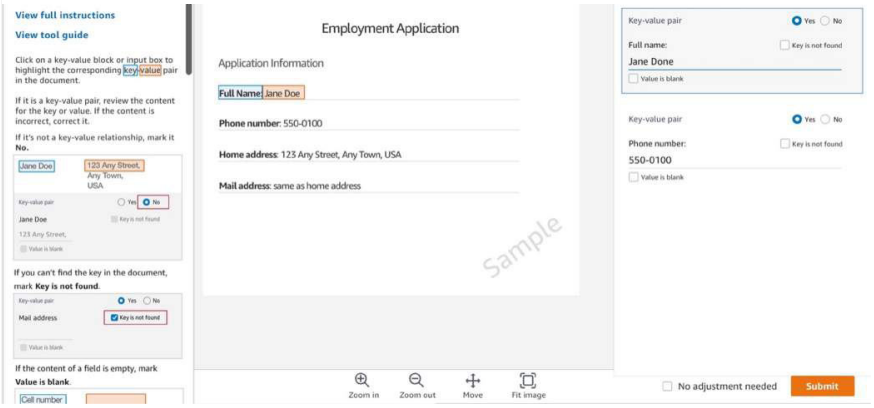
Corgi

Chihuahua

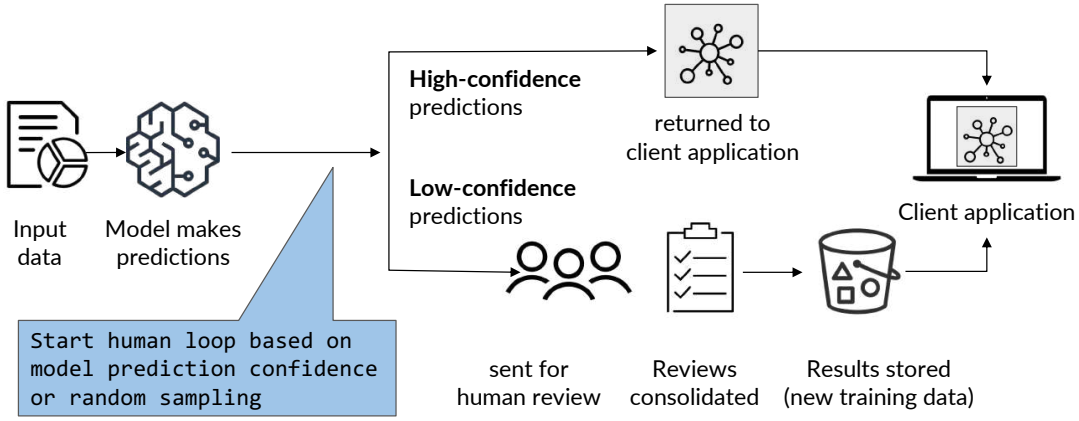


Form Extraction

Task: Review the key-value pairs to the right and correct them if they do not match the document



Implement Human Review Of Model Predictions



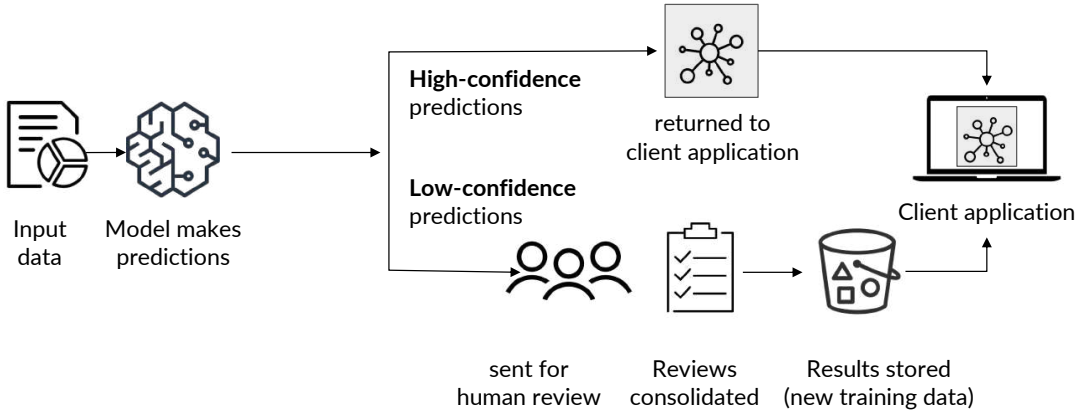
Challenges

- Need ML scientists, engineering, and operations teams
- Need to manage large number of reviewers
- Need to write custom software to manage review tasks
- Difficult to achieve high review accuracy

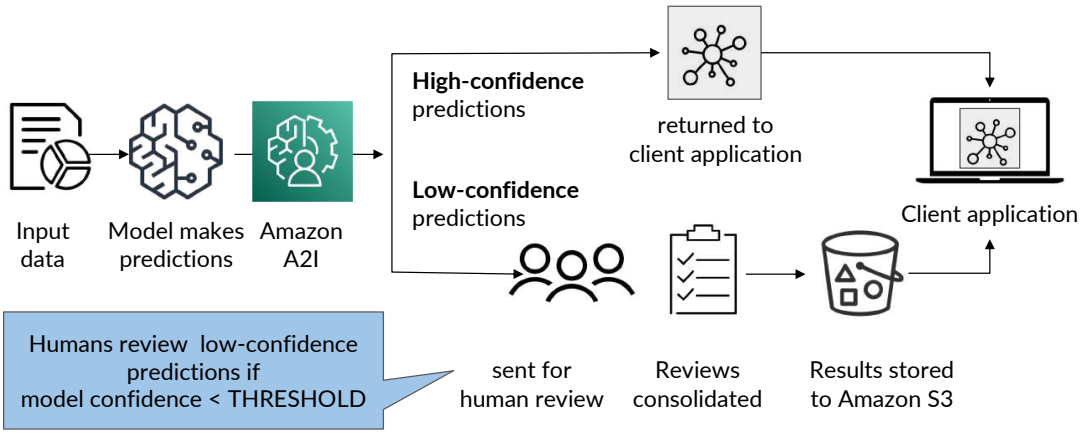
Human-In-The-Loop Pipelines

with Amazon Augmented AI (Amazon A2I)

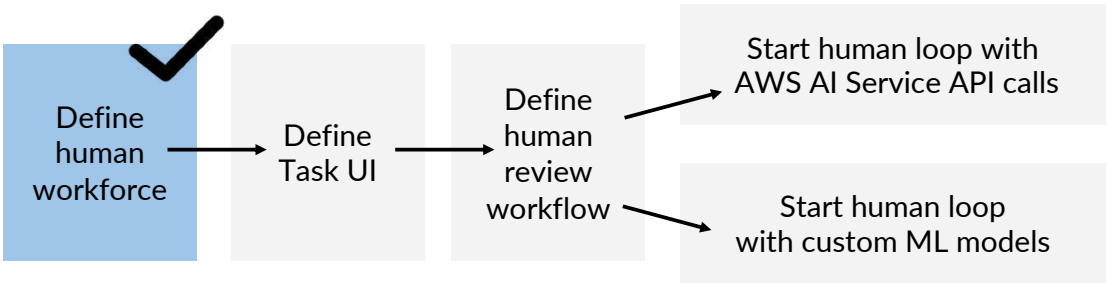
Implement Human Review Of Model Predictions



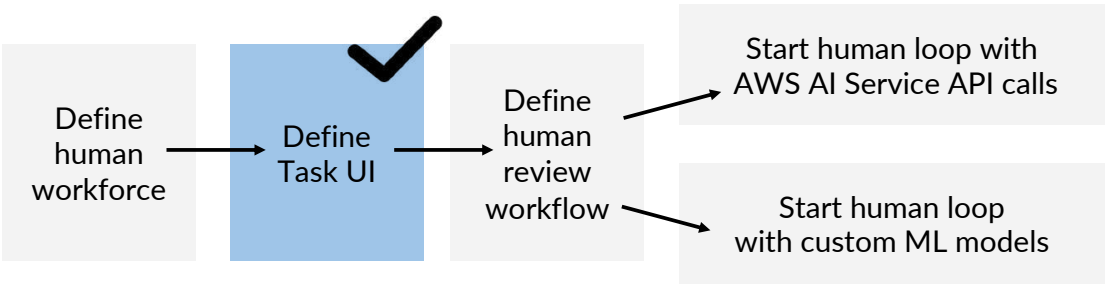
Amazon A2I



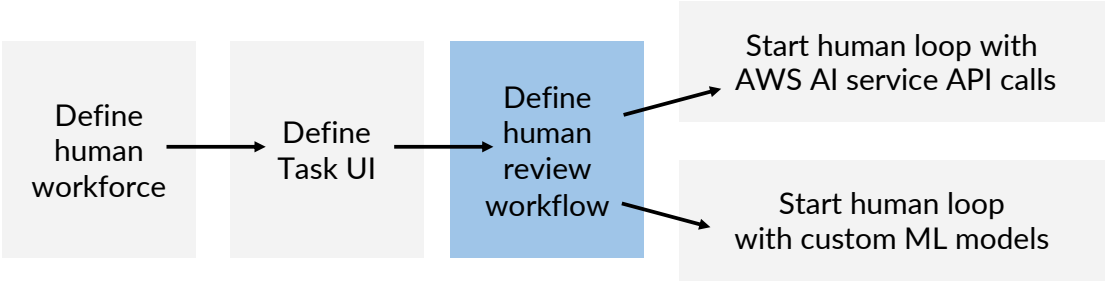
Steps



Steps



Steps



Define Human Review Workflow

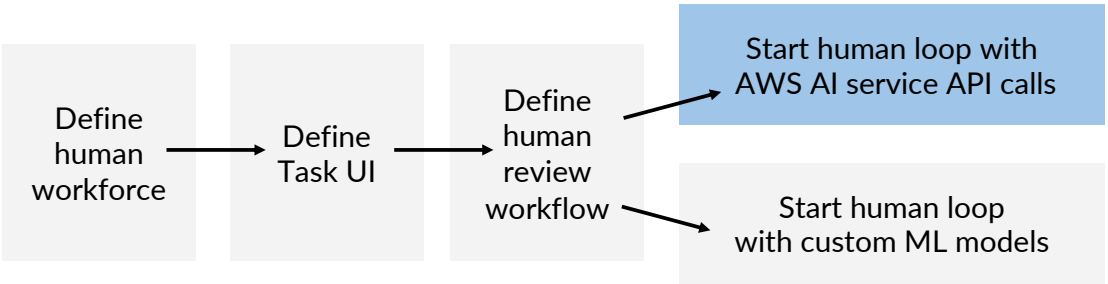
```
create_workflow_definition_response = sm.create_flow_definition(
    FlowDefinitionName=<NAME>,
    RoleArn=role,
    HumanLoopConfig={
        "WorkteamArn": ...,
        "HumanTaskUiArn": ...,
        "TaskCount": 1,
        "TaskDescription": "Classify Reviews into sentiment:  -1 (negative),
                                                    0 (neutral), 1
(positive)",
        "TaskTitle": ... },
    OutputConfig={"S3OutputPath": output_path}, ... )

augmented_ai_flow_definition_arn =
          create_workflow_definition_response["FlowDefinitionArn"]
```

Workforce and team to use

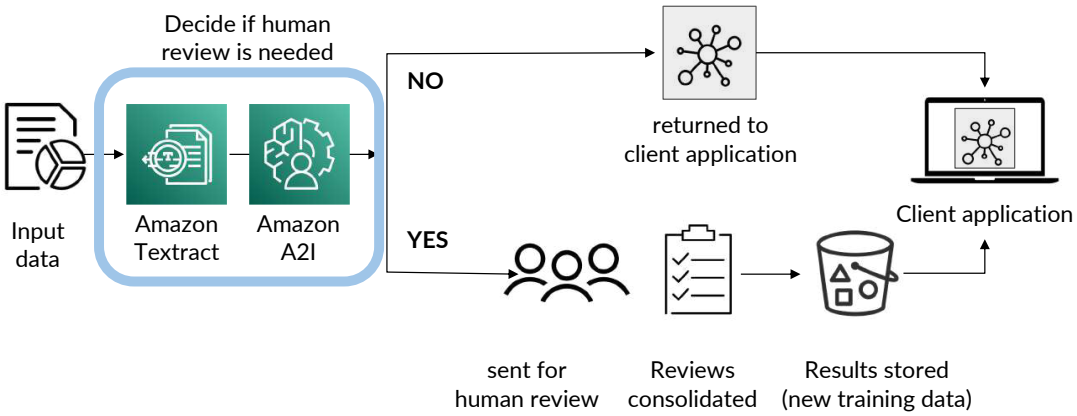
Human Task UI and additional configuration

Where to store output data

Steps



Start Human Loop With Amazon Textract

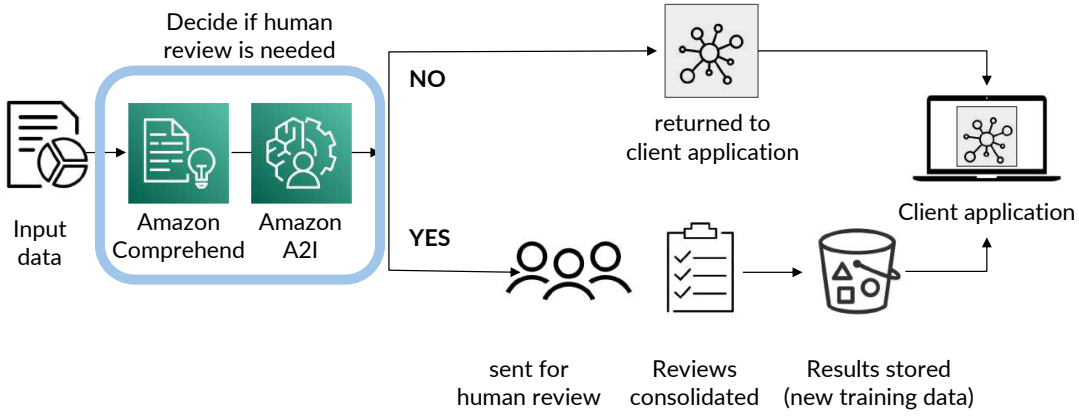


Start Human Loop With Amazon Textract

```
def analyze_document_with_a2i(document_name, bucket):
    response = textract.analyze_document(
        Document={'S3Object': {'Bucket': bucket, 'Name': document_name}},
        ...
        HumanLoopConfig=humanLoopConfig
    )
    return response

analyzeDocumentResponse = analyze_document_with_a2i(document, S3_BUCKET)
```

Add your human loop configuration to the Amazon Textract API call

Start Human Loop With Amazon Comprehend



Start Human Loop With Amazon Comprehend

```
reviews = ["I enjoy this product", "It is okay"]
CONFIDENCE_SCORE_THRESHOLD = 0.90

for review in reviews:
    # Call the a Amazon Comprehend Custom model
    response = comprehend.classify_document(
                                    Text=review,
                                    EndpointArn=comprehend_endpoint_arn)

    sentiment = response["Classes"][0]["Name"]
    confidence_score = response["Classes"][0]["Score"]
    print(f'Processing sample_review: "{review}"')
    ...
```

Sample reviews and confidence sore threshold

Get prediction response from a custom Amazon Comprehend model

Parse sentiment class and confidence score from model response

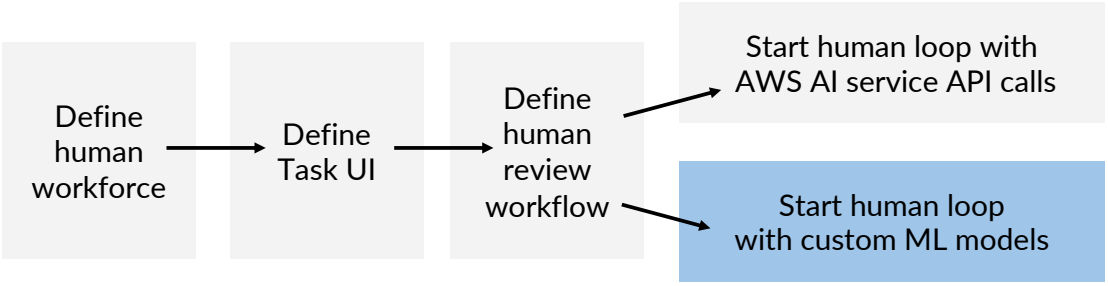
Start Human Loop With Amazon Comprehend

```
    ...
    # Condition when to engage a human for review
    if confidence_score < CONFIDENCE_SCORE_THRESHOLD:
        humanLoopName = <NAME>
        inputContent = {"initialValue": sentiment, "taskObject": sample_review}


        start_loop_response = a2i.start_human_loop(
            HumanLoopName=humanLoopName,
            FlowDefinitionArn=augmented_ai_flow_definition_arn,
            HumanLoopInput={"InputContent": json.dumps(inputContent)},
        )
```

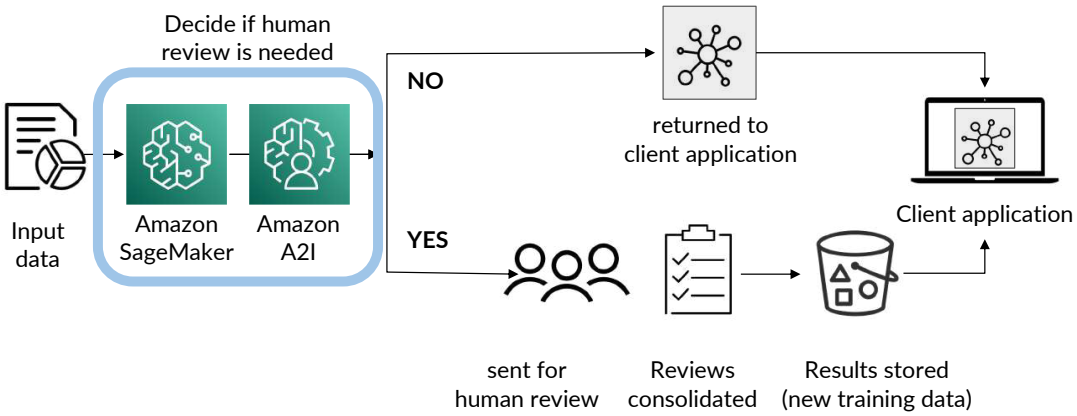
Check returned confidence score

Start Amazon A2I human loop with inputs

Steps



Use Amazon A2I With Amazon SageMaker



Use Amazon A2I With Amazon SageMaker

```
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=...,
                         serializer=...,
                         deserializer=...)
```

Specify how to process model inputs (serializer) and model outputs (deserializer).

Use Amazon A2I With Amazon SageMaker

```
from sagemaker.pytorch.model import PyTorchModel
pytorch_model_name = 'model-{}'.format(timestamp)

model = PyTorchModel(name=...,
                                        predictor_cls=...,
                     model_data='s3://.../model.tar.gz',
                     ...)

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

predictor = model.deploy(endpoint_name=pytorch_endpoint_name, ...)
```

Specify custom predictor

S3 location of model artifact

Deploy model endpoint

Use Amazon A2I With Amazon SageMaker

```
reviews = ["I enjoy this product", "It is okay"]

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

    response = predictor.predict(inputs)

    prediction = response[0]['predicted_label']
    confidence_score = response[0]['probability']
    ...
```

Sample reviews

Call the model through the predictor.predict()

Parse predicted label and confidence score from model response

Use Amazon A2I With Amazon SageMaker

```
    ...
    if confidence_score < CONFIDENCE_SCORE_THRESHOLD:

        human_loop_name = <NAME>
        input_content = {"initialValue": ..., "taskObject": ...}

        start_loop_response = a2i.start_human_loop(
            HumanLoopName=human_loop_name,
            FlowDefinitionArn=augmented_ai_flow_definition_arn,
            HumanLoopInput={"InputContent": json.dumps(input_content)},
        )
         ...
```

Check returned confidence score

Start Amazon A2I human loop with inputs

Human Loops Started

```
[{'probability': 0.937, 'predicted_label': 1}]
Checking prediction confidence 0.937 for sample review: "I enjoy this product"
Confidence score of 93.7% for star rating of 1 is above threshold of 90.0%
No human loop created.

[{'probability': 0.542, 'predicted_label': 1}]
Checking prediction confidence 0.542 for sample review: "It is okay"
Confidence score of 54.2% for prediction of 1 is less than the threshold of 90.0%
*** ==> Starting human loop with name: 1620962189-045527
```


Verify Results

input_content	human_answer
{'initialValue': 1, 'taskObject': 'It is okay'}	{'sentiment': {'label': '0'}}

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.

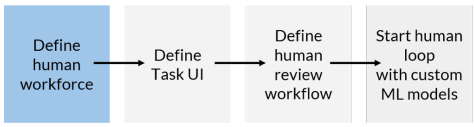
In [2]:
```
# please ignore warning messages during the installation
!pip install --disable-pip-version-check -q sagemaker==2.35.0
```

```
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.



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]:
```
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]:
```
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]:
```
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))
```

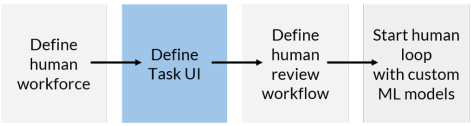
```
{'ResponseMetadata': {'HTTPHeaders': {'connection': 'keep-alive',
                                      'content-length': '242',
                                      'content-type': 'application/x-amz-
json-1.1',
                                      'date': 'Tue, 06 Sep 2022 04:57:16
GMT',
                                      '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



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
```

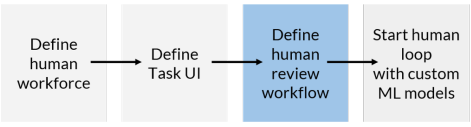
Out[28]:
```
{'HumanTaskUiArn': 'arn:aws:sagemaker:us-east-1:395611198364:human-task-u
i/ui-1662439559',
 'ResponseMetadata': {'RequestId': 'd4fe3452-949a-4e09-86ae-78afad5b34f8'
,
  'HTTPStatusCode': 200,
  'HTTPHeaders': {'x-amzn-requestid': 'd4fe3452-949a-4e09-86ae-78afad5b34
f8',
   'content-type': '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



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.

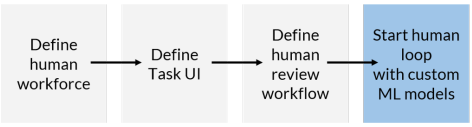
In [31]:
```
# Flow definition name - this value is unique per account and region
flow_definition_name = 'fd-{}'.format(timestamp)

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`.

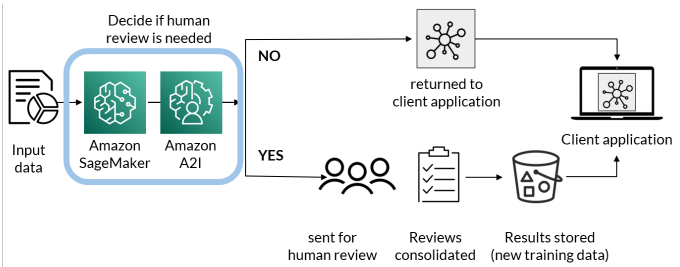
In [32]:
```
for _ in range(60):
    describe_flow_definition_response = sm.describe_flow_definition(FlowD
    print(describe_flow_definition_response["FlowDefinitionStatus"])
    if describe_flow_definition_response["FlowDefinitionStatus"] == "Acti
        print("Flow Definition is active")
       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

```
Active
Flow Definition is active
```

4. Start human loop with custom ML model



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]:
```
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.

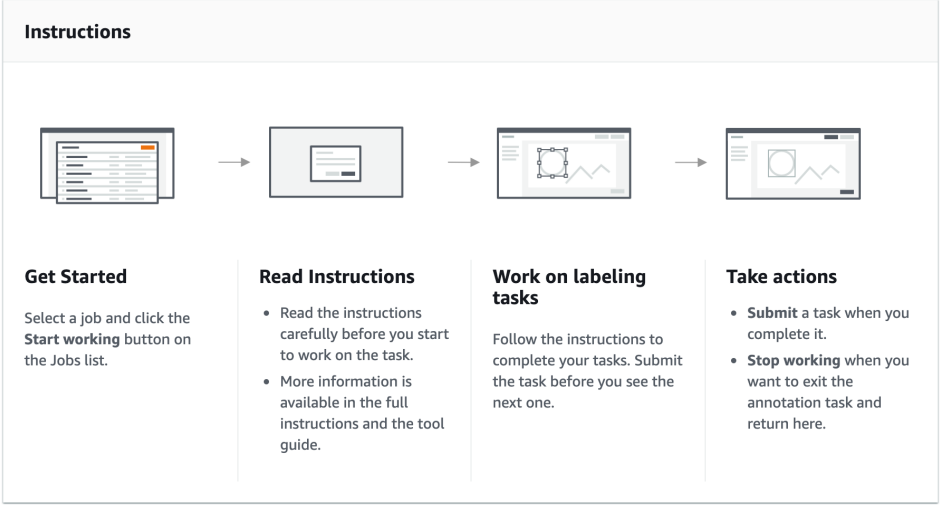
In [40]:
```
labeling_ui = sm.describe_workteam(WorkteamName=workteam_name)["Workteam"
print(labeling_ui)
```

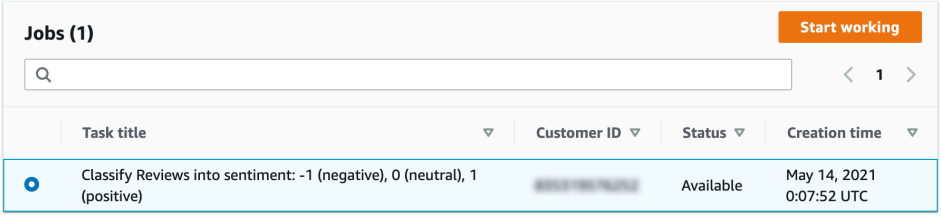
```
4gb4gzwziq.labeling.us-east-1.sagemaker.aws
```

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**





*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]:
```
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]:
```
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 [ ]: