Unleashing the Power of AutoML

Gerald Friedland
UC Berkeley, Amazon Web Services

Debanjan Datta
Amazon Web Services

Introduction

What is AutoML?

AutoML

- *AutoML is defined as a set of methods and processes that are designed to make Machine Learning accessible in a useful but efficient manner for non-Machine Learning experts so that ML can be leveraged for their tasks.*

Abstracting Complexity

Introduction

- Implementing ML system starts with:
  - Selecting and Provisioning a machine
  - Setting up an environment

- Multimedia applications require Deep Learning
  - Specific hardware viz. Cudahy enabled GPU
  - Correct Libraries
  - Correct development environment settings

Practicum: Set up the environment

https://github.com/ddatta-DAC/ACM_MM_2024_Tutorial

- Full Setup
- Quick set up
- Jupyter Lab

Problem Statement

- Images : Key modality in multimedia
- Related ML problem : Image Classification
- Question: How does AutoML help simplify the solution ?

Abstraction of Complexity

Practicum: Download Data

- CIFAR-10 dataset
  - Simple to understand
  - Public
  - We use a subsample (for ease of demonstration)

Notebook

Practicum

- A approach in AutoML: Transfer Learning
  - Pre-trained models can be finetuned on target datasets
  - Sot performance
  - Less computationally expensive
- We train a *Image classification* model
  - Architecture: *ResMed*
  - Train only the final layer of the neural network

Code

Complexity

- Importing correct libraries & dependencies
- Obtaining correct model weights

```
model = torch.hub.load("pytorch/vision:v0.10.0", "resnet18", weights="ResNet18_Weights.DEFAULT")
for param in model.parameters():
    param.requires_grad = False

fc = list(model.children())[-1:]
inp_features = fc[0].in_features
model.fc = Linear(inp_features, num_classes)
```

```
import torch
import os
import sys
from glob import glob
from typing import *

import lightning as pl
import numpy as np
import pandas as pd
import PIL
import torch
import torchvision
import torchvision.transforms as transforms
import torchvision.transforms.functional as F
from matplotlib import pyplot as plt
from matplotlib.pyplot import imshow
from PIL import Image
from torch.nn import Linear, Sequential
from torch.nn import functional as TF
from torch.optim import lr_scheduler
from torch.utils.data import DataLoader, Dataset
from torchmetrics.classification import MulticlassAccuracy
from torchvision import datasets
from torchvision.datasets import ImageFolder
from torchvision.io import read_image
from tqdm.auto import tqdm
from transformers import AutoImageProcessor, ResNetModel
from torch.optim import SGD, AdamW
from colorama import Fore, Back, Style
from time import time
```

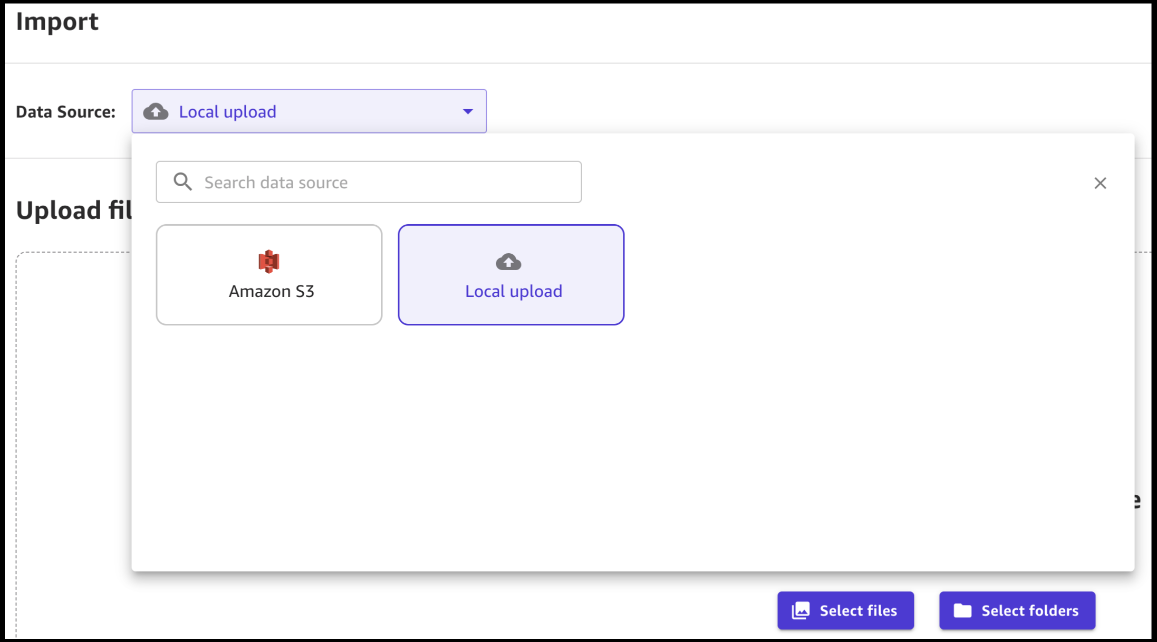
Complexity

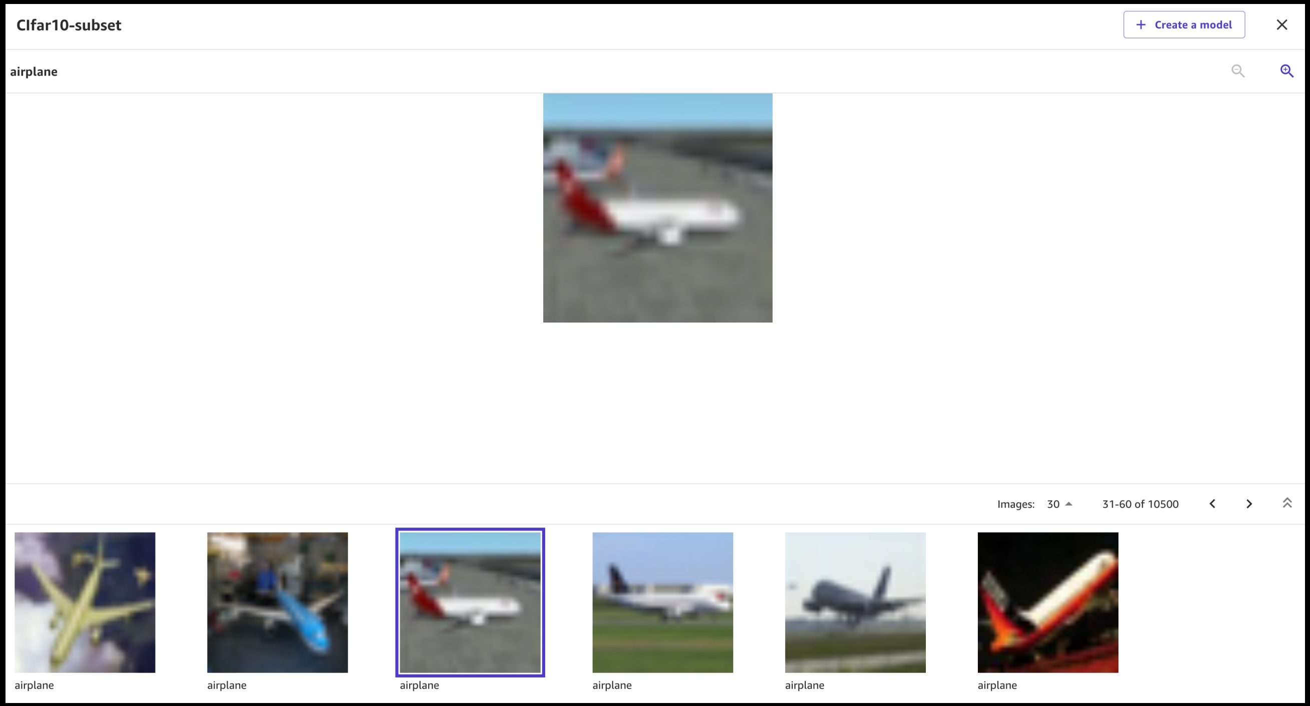
- Model & dataset specific transforms
- Converting data to ML library compatible format e.g. *tensors*

```
transform = transforms.Compose(
    [
        transforms.Resize(
            [224, 224]
        ),  # Resizing the image as the VGG only take 224 x 244 as input size
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ]
)
```

AutoML with AWS SageMaker Canvas

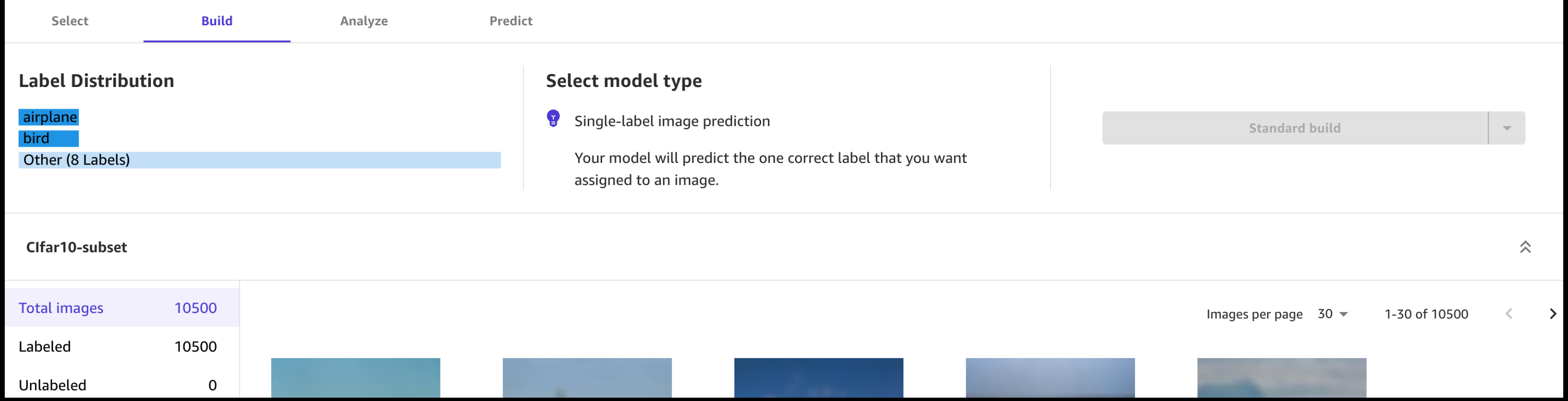
- Create dataset
  - Import from a local source
  - Import from Amazon S3





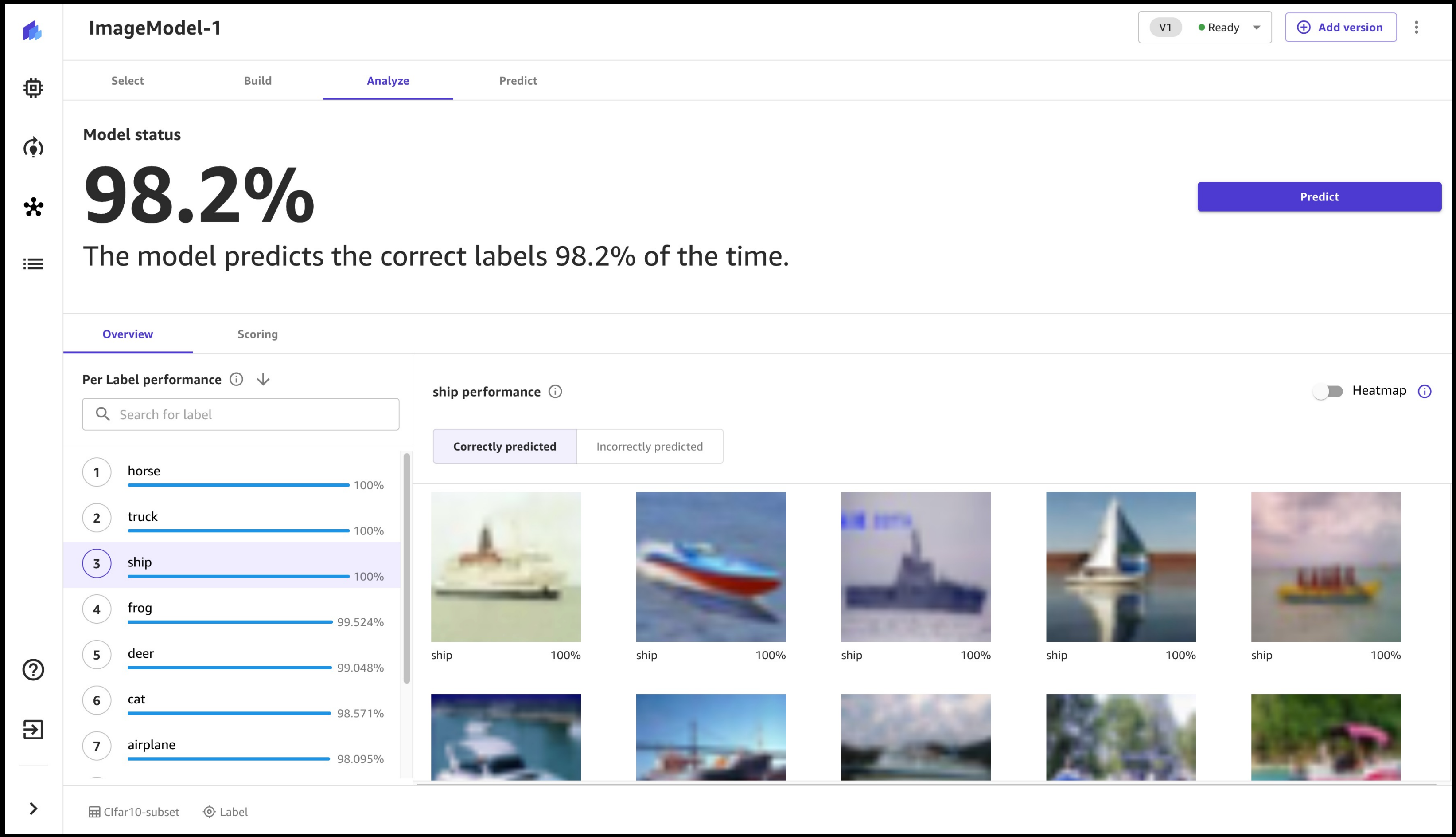
AutoML with AWS SageMaker Canvas

- Visualize the data distribution of labels.
- Select type of model to be built:
  - Quick Build
  - Standard Build



AutoML with AWS SageMaker Canvas

- Single click and the model is built!



Takeaways

- Demonstrate the complexity:
  - Loading data, formatting data
  - Applying transforms
  - Implementing training from scratch
  - Setting training hyper-parameters

- AutoML reduces complexity leading to a cleaner solution!

Hyperparameter & Model Selection

Introduction

- Hyperparameters --- key to ML model performance
- Controls the learning process
- Not Learnt --- unlike parameters of a ML model
    - Needs to be explicitly by user
- Affects the outcome significantly

Hyperparameters

- Difficult to estimate correct values
- Different combinations of hyperparameter
    - Different hyperparameter ranges, types
    - Possibly exponential search space
- Hyperparameters should account for hardware
- Rule of thumb values are not adequate

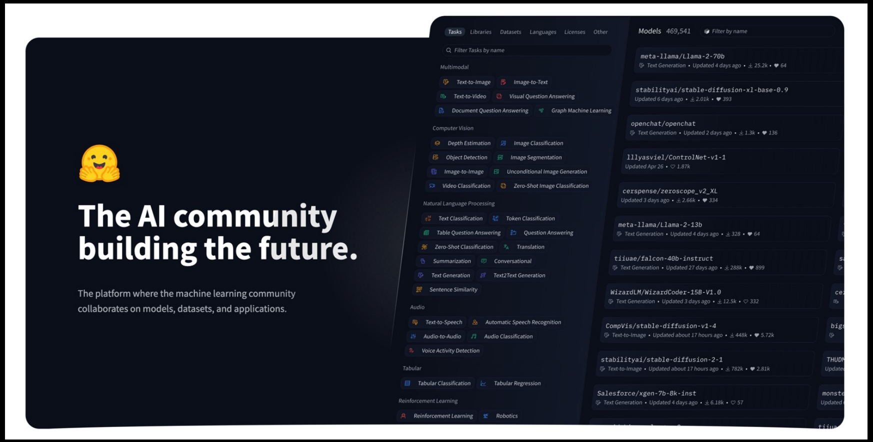
Problem Statement

- A key multimedia modality --- Text
- Related ML problem : Text Classification
- Question : How does AutoML simplifies the solution ?

Model selection & Hyperparameter settings

Problem Statement

- Open source ML solutions : [Huggingface]
- Comparatively simple:
  - Experiment with models
  - Python Libraries with documentation



Practicum

- Data:
  - IMDB Reviews (Open source)
  - Github Link to for data pre-processing and exploration

- Perform Text Classification in *Python*, using ML Libraries
  - Huggingface `transformers`

Code

```
from transformers import (
    AutoModel,
    AutoModelForSequenceClassification,
    AutoTokenizer,
    DataCollatorWithPadding,
    Trainer,
    TrainingArguments,
)
```

Practicum : Effect of Hyperparameters

- Models takes a set of hyperparameters
- `` `model_id` `` : Model

```
args = parser.parse_args()
model_id = args.model_id
gradient_accumulation_steps = args.gradient_accumulation_steps
weight_decay = args.weight_decay
train_batch_size = args.train_batch_size
num_epochs = args.num_epochs
learning_rate = args.learning_rate
```

Practicum: Experiment 1

Experiment & Results

Model	Epochs	Train Batch Size	Learning Rate	Gradient Accumulation Steps	Weight Decay	Train time (seconds)	Accuracy	F1-score
distilbert-base-uncased	2	64	0.0001	1	0.01	33	0.91	0.908722
distilbert-base-uncased	1	64	0.0001	1	0.01	18	0.888	0.883333
roberta-base	2	16	0.05	1	0.10	108	0.518	0.0
distilbert-base-uncased	10	32	0.0002	1	0.01	197	0.886	0.880503
bert-base-uncased	5	16	0.0001	3	0.01	243	0.94	0.939759
microsoft/deberta-base	3	8	5e-05	2	0.01	338	0.958	0.957055

Practicum: Experiment 2

Experiments & Results

Model	Epochs	Train Batch Size	Learning Rate	Gradient Accumulation Steps	Weight Decay	Train time (seconds)	Accuracy	F1-score
distilbert-base-uncased	4	32	0.0002	1	0.005	100.36	0.932	0.92827
distilbert-base-uncased	4	32	0.0002	1	0.01	101.55	0.926	0.92178
distilbert-base-uncased	4	32	0.0002	1	0.25	100.85	0.916	0.91139
distilbert-base-uncased	4	32	0.0002	1	0.5	101.39	0.916	0.91429

Practicum: Experiment 3

Experiments & Results

Model	Epochs	Train Batch Size	Learning Rate	Weight Decay	Train time (seconds)	Accuracy	F1 score
distilbert-base-uncased	5	64	0.0001	0.01	77.58615303039551	0.938	0.936082474226804
distilbert-base-uncased	4	64	0.0001	0.01	62.806885957717896	0.926	0.9227557411273486
distilbert-base-uncased	3	64	0.0001	0.01	47.86117625236511	0.932	0.9282700421940928
distilbert-base-uncased	2	64	0.0001	0.01	33.35843515396118	0.91	0.9087221095334684
distilbert-base-uncased	1	64	0.0001	0.01	18.711568355560303	0.888	0.8833333333333333

Practicum: Experiment 4

Experiments & Results

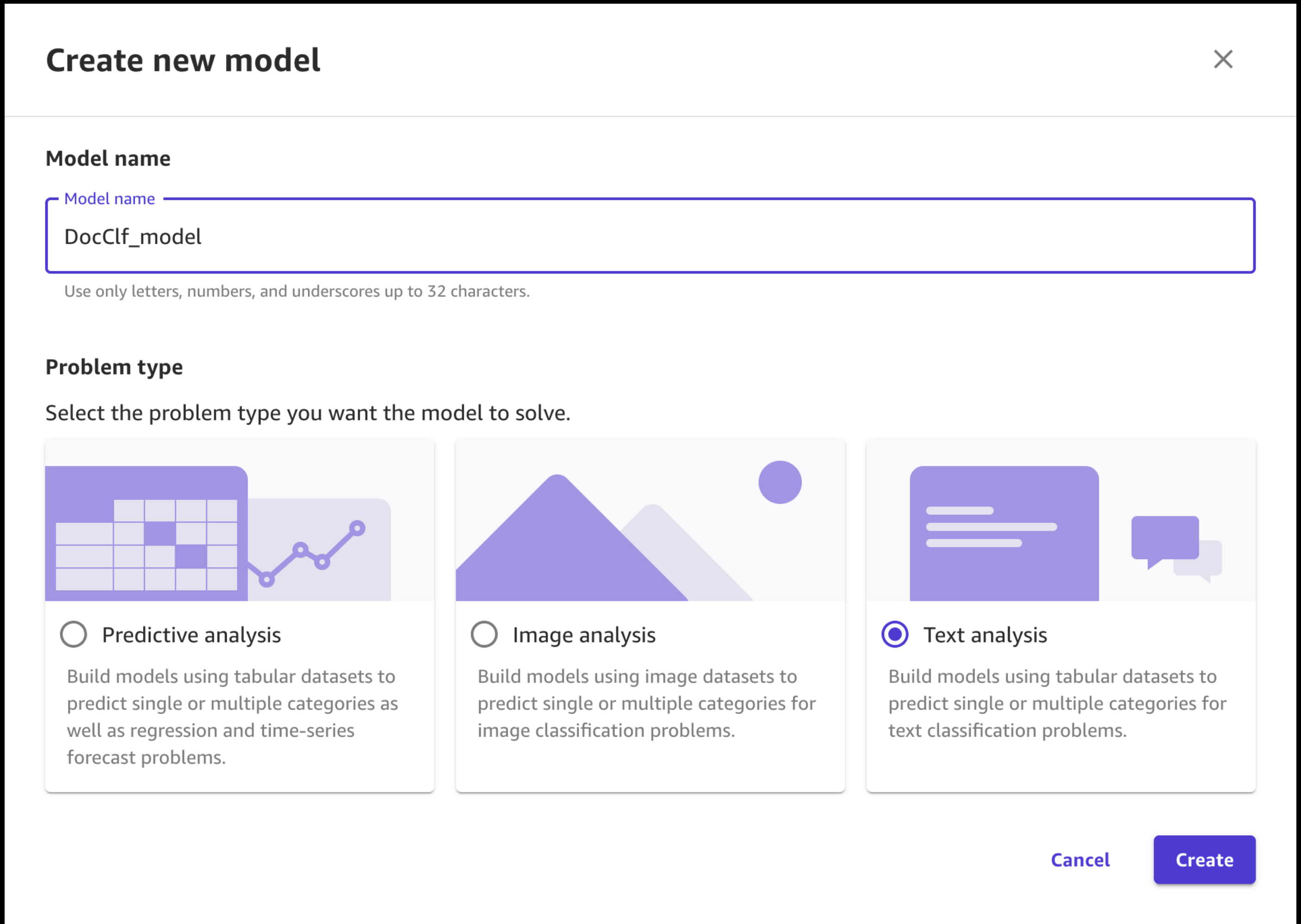
```
microsoft/deberta-base  Epochs: 5LR: 0.0002Weight Decay: 0.01Grad Acc: 1

Downloading (…)okenizer_config.json: 100%| | 52.0/52.0 [00:00<00:00, 13.7kB/s]
Downloading (…)lve/main/config.json: 100%| | 474/474 [00:00<00:00, 168kB/s]
Downloading (…)olve/main/vocab.json: 100%| | 899k/899k [00:00<00:00, 47.2MB/s]
Downloading (…)olve/main/merges.txt: 100%| | 456k/456k [00:00<00:00, 98.7MB/s]
Map: 100%| | 5000/5000 [00:01<00:00, 3942.64 examples/s]
Map: 100%| | 500/500 [00:00<00:00, 4886.18 examples/s]
Map: 100%| | 500/500 [00:00<00:00, 4852.59 examples/s]
Downloading pytorch_model.bin: 100%| | 559M/559M [00:01<00:00, 472MB/s]
Some weights of DebertaForSequenceClassification were not initialized from the model checkpoint at microsoft/deberta-base and are newly initialized: ['pooler.dense.bias', 'pooler.dense.weight', 'classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
  0%|                                                                                      | 0/100 [00:00<?, ?it/s]
You're using a DebertaTokenizerFast tokenizer. Please note that with a fast tokenizer, using the `__call__` method is faster than using a method to encode the text followed by a call to the `pad` method to get a padded encoding.
Traceback (most recent call last):
  File "/home/ubuntu/Code/ACM_MM/ACM_MM_AutoML_Tutorial/Modules/ImageClassification/Module2/textclf.py", line 182, in <module>
    result = main(
  File "/home/ubuntu/Code/ACM_MM/ACM_MM_AutoML_Tutorial/Modules/ImageClassification/Module2/textclf.py", line 139, in main
    trainer.train()
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/transformers/trainer.py", line 1591, in train
    return inner_training_loop(
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/transformers/trainer.py", line 1892, in _inner_training_loop
    tr_loss_step = self.training_step(model, inputs)
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/transformers/trainer.py", line 2776, in training_step
    loss = self.compute_loss(model, inputs)
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/transformers/trainer.py", line 2801, in compute_loss
    outputs = model(**inputs)
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/torch/nn/modules/module.py", line 1518, in _wrapped_call_impl
    return self._call_impl(*args, **kwargs)
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/torch/nn/modules/module.py", line 1527, in _call_impl
    return forward_call(*args, **kwargs)
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/torch/nn/parallel/data_parallel.py", line 185, in forward
    outputs = self.parallel_apply(replicas, inputs, module_kwargs)
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/torch/nn/parallel/data_parallel.py", line 200, in parallel_apply
    return parallel_apply(replicas, inputs, kwargs, self.device_ids[:len(replicas)])
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/torch/nn/parallel/parallel_apply.py", line 110, in parallel_apply
    output.reraise()
  File "/opt/conda/envs/acm_mm/lib/python3.9/site-packages/torch/_utils.py", line 694, in reraise
```

AutoML Solution with AWS SageMaker Canvas

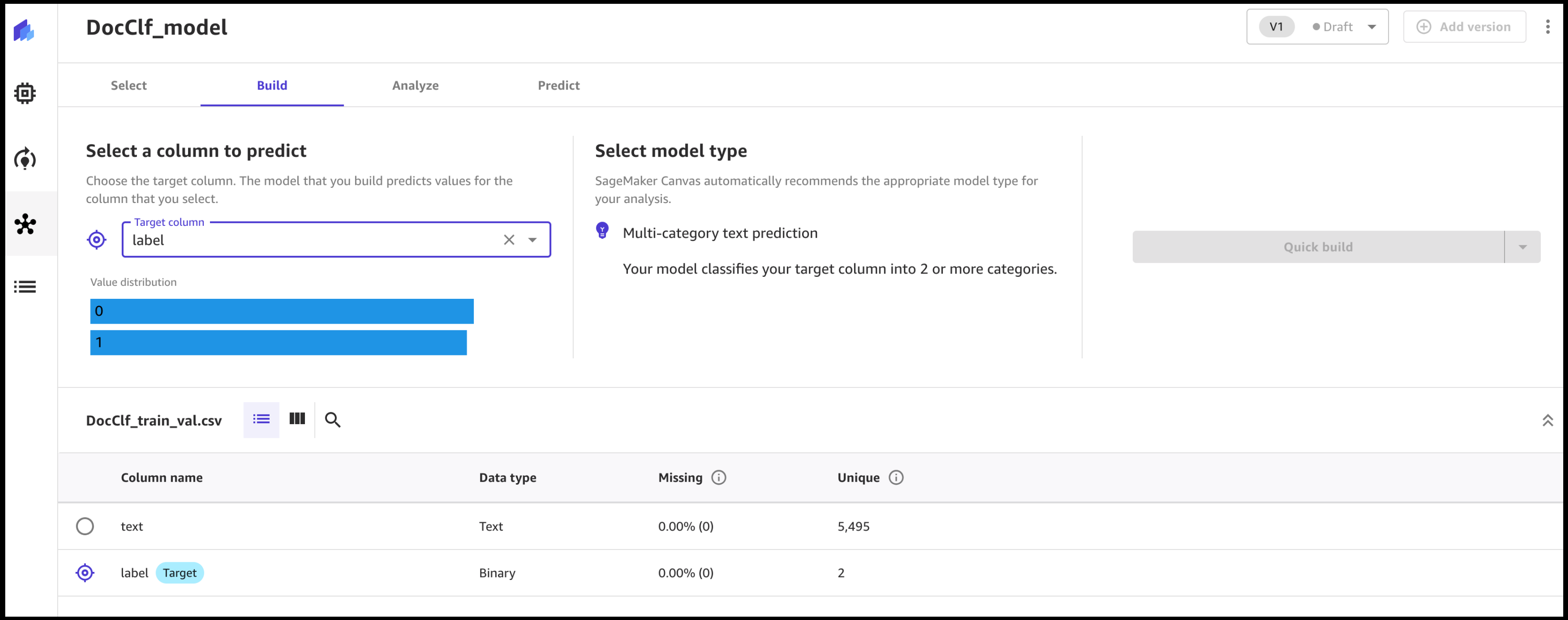
- Few clicks to set up the model
  - Import the data
    - Training + Validation set
    - Test set
  - Select the type of model

- Data accepted in a wide array of standard formats



AutoML Solution with AWS SageMaker Canvas

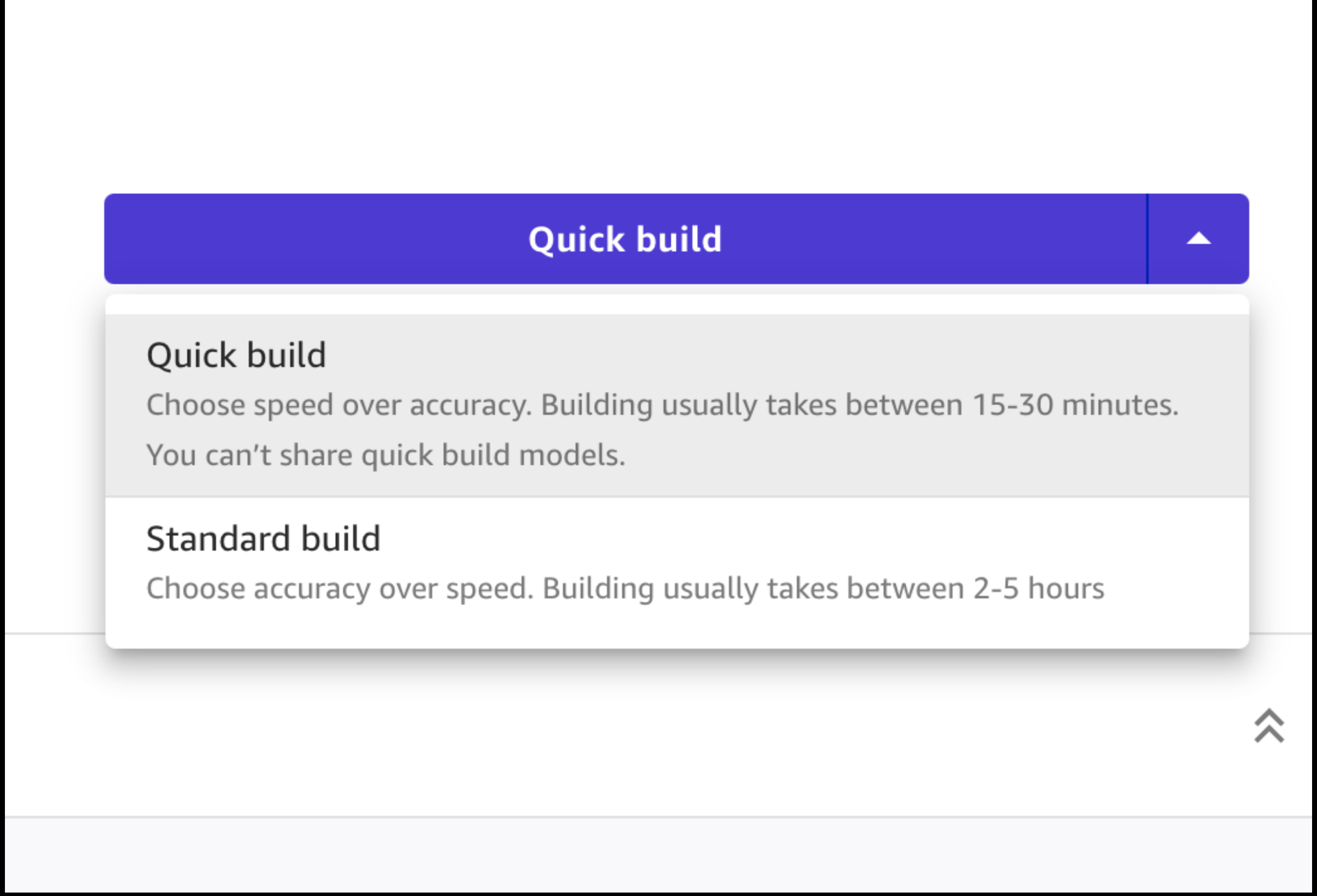
- Visualize data
- Set up the problem



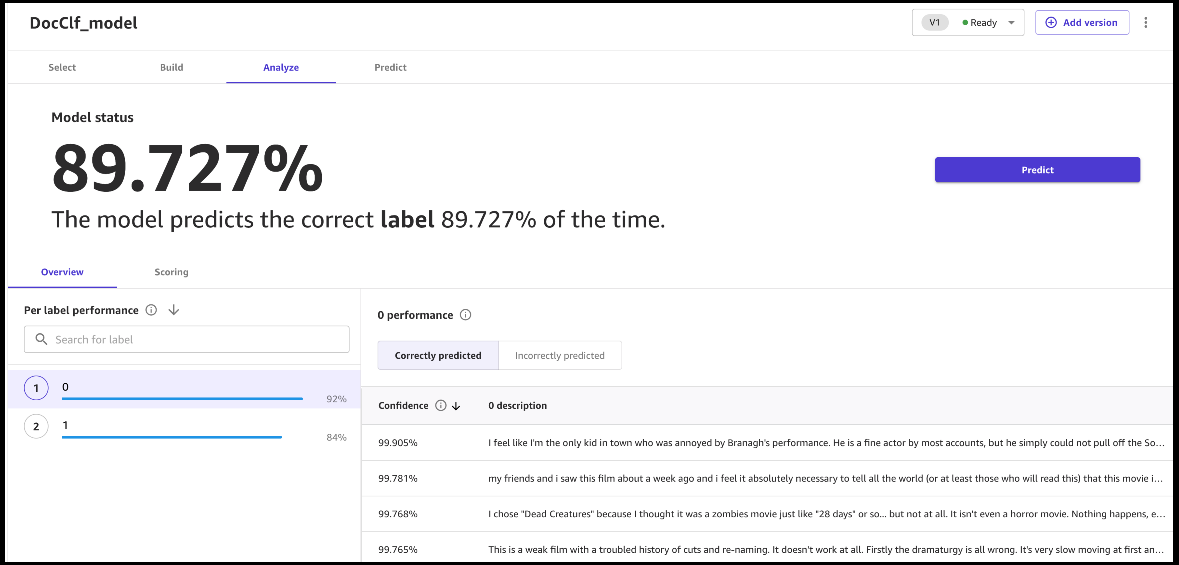
AutoML Solution with AWS SageMaker Canvas

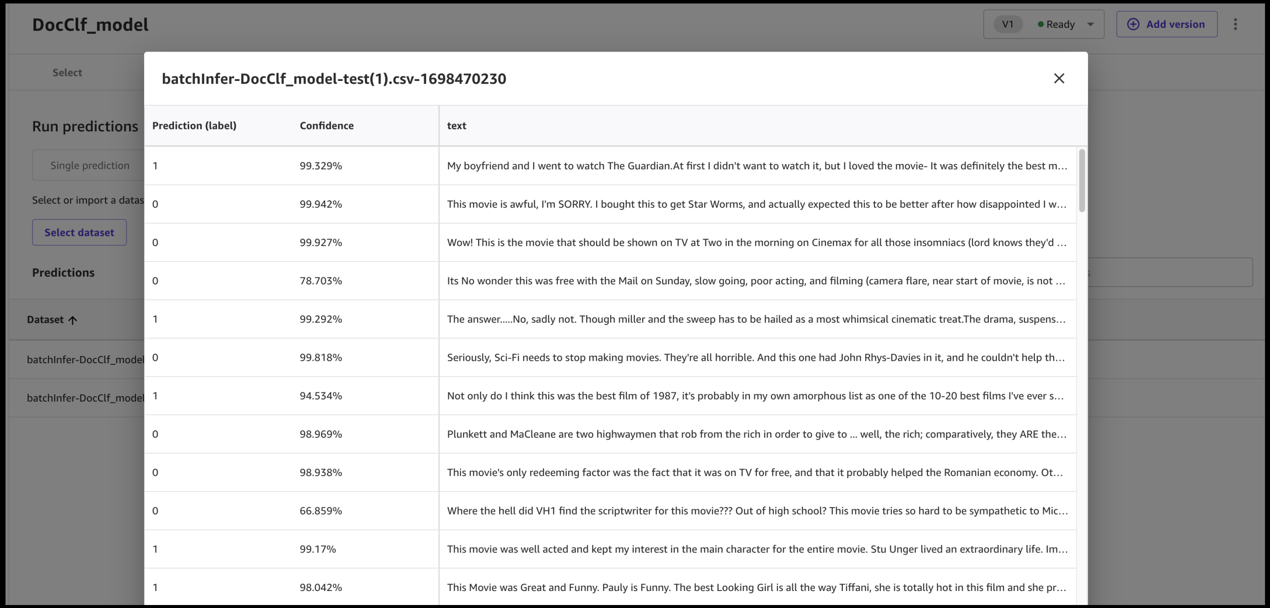
Choose model type.

- Quick build: Fast, but lower performance
  - Quick prototyping
- Standard mode: Higher accuracy, slower.



AutoML Solution with AWS SageMaker Canvas





Takeaways

- Demonstrate the complexity in
  - Understanding and setting training metrics and hyper-parameters

- In summary, Multimedia researchers might spend effort towards obtaining the correct hyperparameters and might have to invest significant time to obtain knowledge on implementing a solution.
- AutoML provides a clean & easy solution.

Conclusion

AWS SageMaker & AutoML

- Canvas enables business analysts and data science teams to build their own models without having to write a single line of code.
- Multimedia types supported are :
  - Text
  - Images
  - Document Images

- AWS SageMaker also provides data preprocessing & manipulation tools that can handle a wide range of datasets.

AWS SageMaker Canvas

Filter by data type: Text Image Document

Last used Grid List

Sentiment analysis

Detect sentiment in lines of text, which can be positive, negative, neutral, or mixed.

Powered by Amazon Comprehend

Expense analysis

Extract information from invoices and receipts, such as date, number, item prices, total amount, and payment terms.

Powered by Amazon Textract

Entities extraction

Extract entities, which are real-world objects such as people, places, and commercial items, or units such as dates and quantities, from text.

Powered by Amazon Comprehend

Language detection

Determine the dominant language in text such as English, French or German.

Powered by Amazon Comprehend

Personal information detection

Detect personal information that could be used to identify an individual, such as addresses, bank account numbers, and phone numbers, from text.

Powered by Amazon Comprehend

Object detection in images

Detect objects, concepts, scenes, and actions in your images.

Powered by Amazon Rekognition

Text detection in images

Detect text in your images.

Powered by Amazon Rekognition

Identity document analysis

Extract information from passports, driver licenses, and other identity documentation issued by the US Government.

Powered by Amazon Textract

Document analysis

Analyze documents and forms for relationships among detected text.

Conclusion

- We demonstrate how AWS Sagemaker provides a comprehensive solution through Canvas & other products
- Researchers can utilize ready-made AutoML solutions to achieve research objectives !

Questions

Gerald Friedland
Principal Applied Scientist, AWS
gfriedla@amazon.com

Debanjan Datta
Applied Scientist, AWS
ddebanja@amazon.com