

Operationalizing Machine Learning

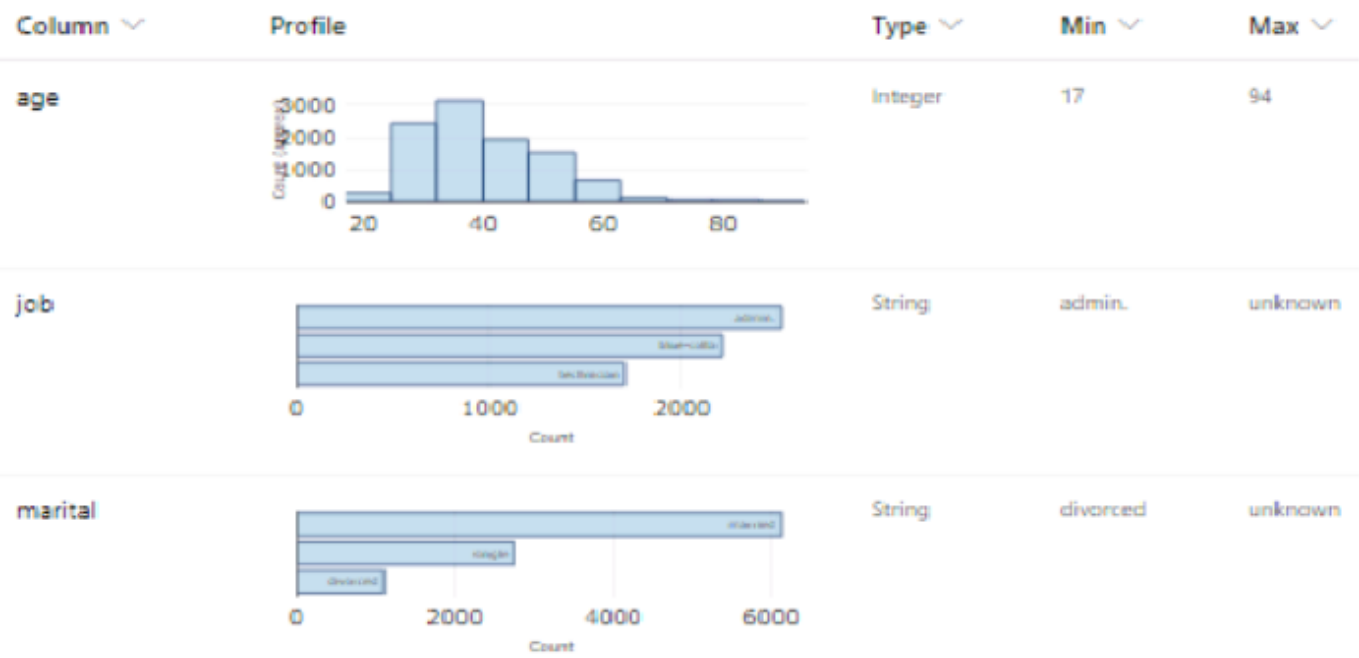
Overview

This project is part of the Udacity Azure ML Nanodegree.

In this project, we use Azure to **configure a cloud-based machine learning production model, deploy it, and consume it**. We will also create, publish, and consume a pipeline.

Summary

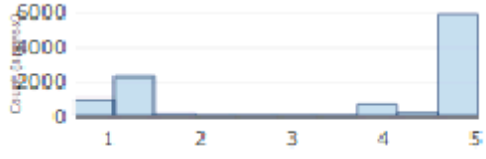

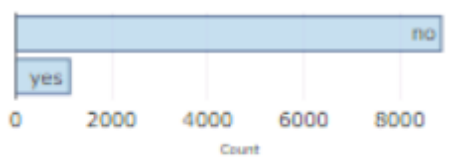
Personal information: Occupation, age, marital status and education.



Financial information: Debts and other data about customer's financial health.

Column ▾	Profile	Type ▾	Min ▾	Max ▾
default		String	no	yes
housing		String	no	yes
loan		String	no	yes

Target: The variable we seek to predict is the one that tells us if a given person is a potential client or not.

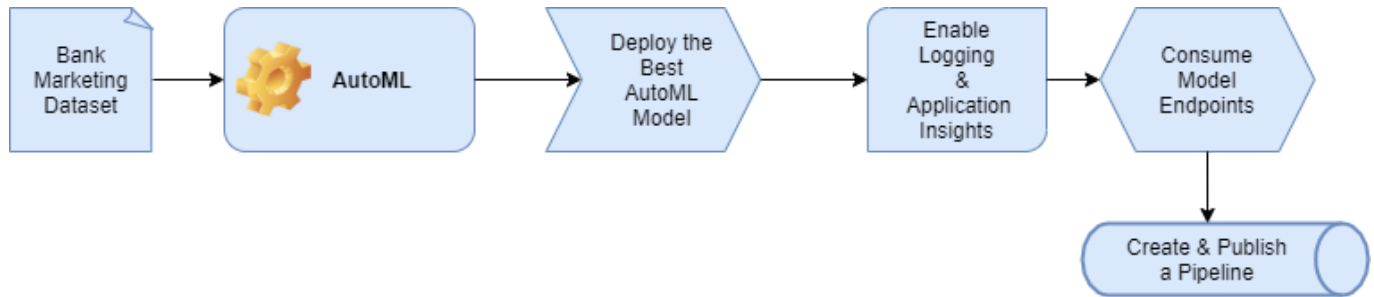
Column ▾	Profile	Type ▾	Min ▾	Max ▾	Count ▾
euribor3m		Decimal	0.63	5.04	10000
nr.employed		Decimal	4963.60	5228.10	10000
y		String	no	yes	10000

Architectural Diagram

In this project, we explain briefly how to publish your best AutoML model and deploy it as a Web API.

- **Experiment Run:** Using MLStudio we manually upload data, select the compute target and the task we want to accomplish
- **Best Model Selection:** Comparing all the models, finally we choose the one with better primary metric, in this case, weighted AUC, according to the target unbalance.
- **Model Deployment:** The best model is deployed as an endpoint.
- **Application Insights Activation:** We enable this logs monitoring tool.

- **Display Swagger Documentation:** We make use of the *swagger.json* file given by Azure to visualize documentation in a more clear and professional way.
- **Consume Endpoint:** We do a test and a benchmark to check out that the endpoint works and to better know latency times.
- **Create, Publish and Consume a Pipeline using Python SDK:** In the top of Automation we have the pipelines. Processing data and retraining only with an HTTP request is possible.



Key Steps

1. **Data Preparation:** When AutoML run is created, *Bank-Marketing* data is uploaded and registered so that we can use it in our experiments.

Microsoft Azure Machine Learning

quick-starts-ws-138938 > Datasets > Bank-marketing

Bank-marketing Version 1 (latest)

Profile: This is the quick profile generated by the top 10,000 rows. Please generate profile to view the schema and summary statistics for full data.

Details Consume Explore Models

Refresh Generate profile Unregister New version

Preview Profile

Number of columns: 21 Number of rows: 50 (of 10000)

	id	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign
1	57	technician	married	highschool	no	no	yes	cellular	may	mon	371	1	
2	55	unknown	married	unknown	unknown	yes	no	telephone	may	thu	285	2	
3	33	blue-collar	married	basic.9y	no	no	no	cellular	may	fri	52	1	
4	36	admin.	married	highschool	no	no	no	telephone	jun	fri	355	4	
5	27	housemaid	married	highschool	no	yes	no	cellular	jul	fri	189	2	
6	58	retired	married	professional.course	no	yes	yes	cellular	jul	fri	605	1	
7	48	services	married	highschool	unknown	yes	no	telephone	may	wed	243	1	

2. **Experiment Run:** AutoML experiment correctly run and submitted.



3. **Best Model:** The best performing model is the one using *VotingEnsemble*. As it was mentioned in **ML-Pipeline** project, due to the high target unbalancement, we're going to focus on *macro* metrics.

Algorithm name	Explained	Accuracy ↓	Sampling ⓘ	Run	Created	Duration	Status
VotingEnsemble		0.91988	100.00 %	Run 82	Feb 15, 2021 8:27 AM	1m 9s	Completed
SparseNormalizer, XGBoostClassifier		0.91563	100.00 %	Run 51	Feb 15, 2021 8:20 AM	1m 3s	Completed
SparseNormalizer, XGBoostClassifier		0.91502	100.00 %	Run 65	Feb 15, 2021 8:23 AM	1m 1s	Completed
SparseNormalizer, XGBoostClassifier		0.91502	100.00 %	Run 62	Feb 15, 2021 8:23 AM	57s	Completed
SparseNormalizer, XGBoostClassifier		0.91472	100.00 %	Run 52	Feb 15, 2021 8:20 AM	56s	Completed
MaxAbsScaler, LightGBM		0.91411	100.00 %	Run 74	Feb 15, 2021 8:25 AM	1m 14s	Completed
SparseNormalizer, XGBoostClassifier		0.91381	100.00 %	Run 68	Feb 15, 2021 8:24 AM	53s	Completed
SparseNormalizer, XGBoostClassifier		0.91351	100.00 %	Run 38	Feb 15, 2021 8:18 AM	1m 5s	Completed

Run 82 Completed

[Refresh](#) [Deploy](#) [Download](#) [Explain model](#) [Cancel](#)

[Details](#) [Model](#) [Explanations \(preview\)](#) [Metrics](#) [Outputs + logs](#) [Images](#) [Child runs](#) [Snapshot](#)

Select a metric to see a visualization or table of the data.

Search

☐ average_precision_score_macro

☐ average_precision_score_micro

☐ average_precision_score_weight...

☐ balanced_accuracy

☒ confusion_matrix

☒ f1_score_macro

☐ f1_score_micro

☐ f1_score_weighted

☐ log_loss

☐ matthews_correlation

☐ norm_macro_recall

☒ precision_score_macro

☐ precision_score_micro

☐ precision_score_weighted

View as: ☒ Chart ☐ Table

AUC_macro

0.946

recall_score_macro

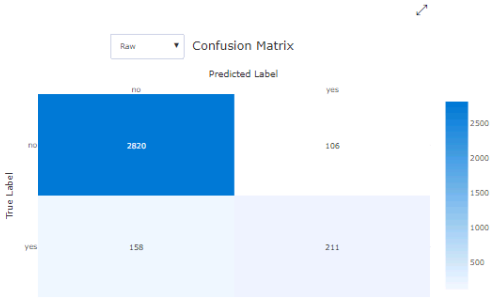
0.768

precision_score_macro

0.806

f1_score_macro

0.785



4. **Deploy Model:** Create an endpoint associated to an ACL.

demo-model-deploy

[Details](#)
[Test](#)
[Consume](#)
[Deployment logs](#)

Service ID

demo-model-deploy

Description

--

Deployment state

Healthy 

Compute type

ACI

Created by

ODL_User 139278

Model ID

[AutoML88152109066:1](#)

Created on

2/21/2021 6:02:58 PM

Last updated on

2/21/2021 6:02:58 PM

Image ID

--

REST endpoint

<http://66f1bb41-2922-4b94-ae09-fab5a70c7d3e.southcentralus.azurecontainer.io/score>

Key-based authentication enabled

true

Swagger URI

<http://66f1bb41-2922-4b94-ae09-fab5a70c7d3e.southcentralus.azurecontainer.io/swagger.json>

5. Activate Application Insights:

1. Check out az extension is installed with `az version` and `az extension add -n azure-cli-ml`.
2. Create a python virtual environment with `virtualenv venv`.
3. Edit and run `logs.py` writing the endpoint name (`demo-model-deploy` in this case) and check output in console:

```
PS C:\Users\demouser\Desktop\nd00333_AZML\02-master\starter_files> python logs.py
Performing interactive authentication. Please follow the instructions on the terminal.
WARNING: Note, we have launched a browser for you to login. For old experience with device code, use "az login --use-device-code"
You have logged in. Now let us find all the subscriptions to which you have access...
Interactive authentication successfully completed.
2021-02-21T18:06:38.2000455Z+00:00 - pyslog/run
2021-02-21T18:06:38.209681684+00:00 - iot-server/run
2021-02-21T18:06:38.209512882+00:00 - gunicorn/run
2021-02-21T18:06:38.212645846+00:00 - nginx/run
/usr/sbin/nginx: /azureml-envs/azureml_09ff5f546b313bb1ab136466214499/lib/libcrypto.so.1.0.0: no version information available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_09ff5f546b313bb1ab136466214499/lib/libcrypto.so.1.0.0: no version information available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_09ff5f546b313bb1ab136466214499/lib/libssl.so.1.0.0: no version information available (required by /usr/sbin/nginx)
/usr/sbin/nginx: /azureml-envs/azureml_09ff5f546b313bb1ab136466214499/lib/libssl.so.1.0.0: no version information available (required by /usr/sbin/nginx)
pyslogds: /azureml-envs/azureml_09ff5f546b313bb1ab136466214499/lib/libbuild.so.1: no version information available (required by pyslogds)
OpenConnectionString and IDENTIFY_IDTIDNAME are not set. Exiting...
2021-02-21T18:06:38.309989575+00:00 - iot-server/finish 1 0
2021-02-21T18:06:38.31100205+00:00 - Exit code 1 is normal. Not restarting iot-server.
Starting gunicorn 35.9.0
Listening at http://127.0.0.1:31311 (12)
Using worker: sync
worker timeout is set to 300
Booting worker with pid: 43
SPARK_HOME not set. Skipping PySpark Initialization.
Generating new frontManager, this may take some time...
Initializing logger
2021-02-21 18:06:39.858 | root | INFO | Starting up app insights client
2021-02-21 18:06:39.858 | root | INFO | Starting up request id generator
2021-02-21 18:06:39.858 | root | INFO | Starting up app insight hooks
2021-02-21 18:06:39.858 | root | INFO | Invoking user's init function
2021-02-21 18:06:42.826 | root | INFO | User's init has completed successfully
2021-02-21 18:06:42.830 | root | INFO | Skipping middleware: dbg_model_info as it's not enabled.
2021-02-21 18:06:42.830 | root | INFO | Skipping middleware: dbg_resource_usage as it's not enabled.
2021-02-21 18:06:42.832 | root | INFO | Scoring timeout is round from os.getenv: 60000 ms
2021-02-21 18:06:47.764 | root | INFO | 200
127.0.0.1 - - [21/Feb/2021:18:06:47 +0000] "GET /swagger.json HTTP/1.0" 200 3258 "-" "Go-http-client/1.1"
2021-02-21 18:06:55.655 | root | INFO | 200
127.0.0.1 - - [21/Feb/2021:18:26:59 +0000] "GET /swagger.json HTTP/1.0" 200 3258 "-" "Go-http-client/1.1"
2021-02-21 18:29:25.205 | root | INFO | 200
127.0.0.1 - - [21/Feb/2021:18:29:29 +0000] "GET /swagger.json HTTP/1.0" 200 3258 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.163 Safari/537.36 Edg/80.0.361.111"
2021-02-21 18:30:25.374 | root | INFO | 200
127.0.0.1 - - [21/Feb/2021:18:30:23 +0000] "GET /swagger.json HTTP/1.0" 200 3258 "-" "Go-http-client/1.1"
2021-02-21 18:30:05.012 | root | INFO | 200
127.0.0.1 - - [21/Feb/2021:18:39:05 +0000] "GET /swagger.json HTTP/1.0" 200 3258 "-" "Go-http-client/1.1"
2021-02-21 18:39:07.915 | root | INFO | 200
127.0.0.1 - - [21/Feb/2021:18:39:07 +0000] "GET /swagger.json HTTP/1.0" 200 3258 "-" "Go-http-client/1.1"
```

4. Check that **Application Insights** is enabled and working: **demo-model-deploy**

Details Test Consume Deployment logs

Model ID
[AutoML88152109066:1](#)

Created on
2/21/2021 6:02:58 PM

Last updated on
2/21/2021 6:39:07 PM

Image ID
--

REST endpoint
<http://66f1bb41-2922-4b94-ae09-fab5a70c7d3e.southcentralus.azurecontainer.io/score>

Key-based authentication enabled
true

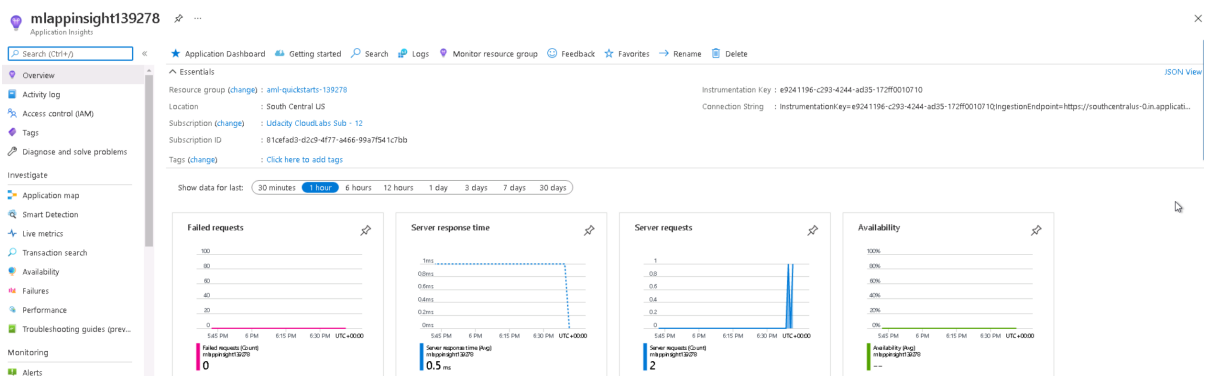
Swagger URI
<http://66f1bb41-2922-4b94-ae09-fab5a70c7d3e.southcentralus.azurecontainer.io/swagger.json>

CPU
1.8

Memory
4 GB

Application Insights enabled
true

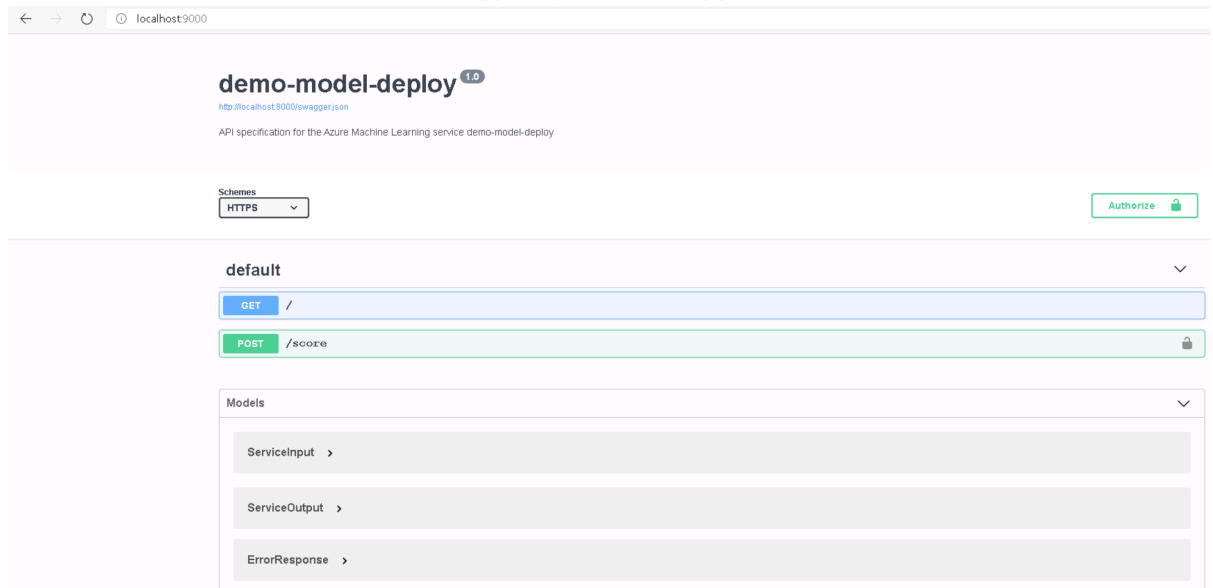
Application Insights url
<https://portal.azure.com#resource/subscriptions/81cefad3-d2c9-4f77-a466-99a7f541c7bb/resourcegroups/aml-quickstarts-139278/providers/microsoftinsights/components/mlappinsight139278>



6. Display Swagger Documentation

1. Download swagger.json from *Details* tab inside the endpoint.
2. Change port from recommended 80 to 9000 in `swagger.sh` because 80 is not available and run it to start the `swagger-ui` docker container. Now it is displayed in `http://localhost:9000/`.
3. Run `server.py` in 8000 port

4. Check <http://localhost:8000/swagger.json> in Swagger.



7. Consume Endpoint and benchmarking

1. Edit *endpoint.py* with the URL and the key necessary to authorize the HTTP request. Some data in a json format is used to make the request and see what the best model predicts from it.

```

GNU nano 4.9.3
import requests
import json

# URL for the web service, should be similar to:
# 'http://8530a665-66f3-49c8-a953-b82a2d312917.eastus.azurecontainer.io/score'
scoring_uri = 'http://66f1bb41-2922-4b94-ae09-fab5a70c7d3e.southcentralus.azurecontainer.io/score'
# If the service is authenticated, set the key or token
key = 'UQTwrNMHnpVlcGdwXWvXdwJNYId9jfw8'

# Two sets of data to score, so we get two results back
data = {"data":
    [
        {
            "age": 17,
            "campaign": 1,
            "cons.conf.idx": -46.2,
            "cons.price.idx": 92.893,
            "contact": "cellular",
            "day_of_week": "mon",
            "default": "no",
            "duration": 971,
            "education": "university.degree",
            "emp.var.rate": -1.8,
            "euribor3m": 1.299,
            "housing": "yes",
            "job": "blue-collar",
            "loan": "yes",
            "marital": "married",
            "month": "may",
            "nr.employed": 5099.1,
            "pdays": 999,
            "poutcome": "failure",
            "previous": 1
        },
        {
            "age": 87,
            "campaign": 1,
            "cons.conf.idx": -46.2,
            "cons.price.idx": 92.893,
            "contact": "cellular",
            "day_of_week": "mon",
            "default": "no",
            "duration": 471,
            "education": "university.degree",
            "emp.var.rate": -1.8,
            "euribor3m": 1.299,
            "housing": "yes",
            "job": "blue-collar",
            "loan": "yes",
            "marital": "married",
            "month": "may",
            "nr.employed": 5099.1,
            "pdays": 999,
            "poutcome": "failure",
            "previous": 1
        }
    ]
}

# Convert to JSON string
input_data = json.dumps(data)
with open("data.json", "w") as _f:
    _f.write(input_data)

# Set the content type
headers = {'Content-Type': 'application/json'}
# If authentication is enabled, set the authorization header
headers['Authorization'] = f'Bearer {key}'

# Make the request and display the response
resp = requests.post(scoring_uri, input_data, headers=headers)
print(resp.json())

PS C:\Users\demouser\Desktop\nd00333_AZMLND_C2-master\starter_files> python .\endpoint.py
{"result": ["yes", "no"]}

```


2. Now we want to know the average time necessary to receive our responses. For that we use **Apache Benchmark**. First of all we check that it is installed displaying the help `ab -h`. Once we see that it works, Authorization key and URI can be introduced into `benchmark.sh` and run the script.

```
GNU nano 4.9.3 benchmark.sh
# HTTP/1.0 200 OK
# Content-Length: 33
# Content-Type: application/json
# Date: Thu, 30 Jul 2020 12:33:34 GMT
# Server: nginx/1.10.3 (Ubuntu)
# X-MS-Request-ID: b48dd8da-0b4e-44fd-ae5-04043bfa77f1
# X-MS-Run-Function-Failed: False
# [{"result": "\yes", "\no"}]
# LOG: Response code = 200
# ..done

# Server Software:      nginx/1.10.3
# Server Hostname:      8530a665-66f3-49c8-a953-b82a2d312917.eastus.azurecontainer.io
# Server Port:          80
# Document Path:        /score
# Document Length:      33 bytes
#
# Concurrency Level:    1
# Time taken for tests:  1.599 seconds
# Complete requests:    10
# Failed requests:      0
# Total transferred:    2600 bytes
# Total body sent:      10560
# HTML transferred:     330 bytes
# Requests per second:  6.25 [#/sec] (mean)
# Time per request:     159.918 [ms] (mean)
# Time per request:     159.918 [ms] (mean, across all concurrent requests)
# Transfer rate:         1.59 [Kbytes/sec] received
#                        6.45 kb/s sent
#                        8.04 kb/s total
#
# Connection Times (ms)
#      min  mean[+/-sd] median  max
# Connect:    21    23   0.8    23    24
# Processing:  52   137  28.3   151   176
# Waiting:    92   137  28.3   151   176
# Total:     114   160  28.0   172  199#

ab -n 10 -v 4 -p data.json -T 'application/json' -H 'Authorization: Bearer UQTwrMhnpVlGdwwvYxwDlNVId9jFW8' http://66f1bb41-2922-4b94-ae09-fab5a70c7d3e.southcentralus.azurecontainer.io/score

Server Software:      nginx/1.10.3
Server Hostname:      66f1bb41-2922-4b94-ae09-fab5a70c7d3e.southcentralus.azurecontainer.io
Server Port:          80
#
Document Path:        /score
Document Length:      33 bytes
#
Concurrency Level:    1
Time taken for tests:  2.269 seconds
Complete requests:    10
Failed requests:      0
Total transferred:    2600 bytes
Total body sent:      10640
HTML transferred:     330 bytes
Requests per second:  4.41 [#/sec] (mean)
Time per request:     226.853 [ms] (mean)
Time per request:     226.853 [ms] (mean, across all concurrent requests)
Transfer rate:         1.12 [Kbytes/sec] received
                        4.58 kb/s sent
                        5.70 kb/s total
#
Connection Times (ms)
      min  mean[+/-sd] median  max
Connect:     2     3   0.5     3     4
Processing: 111   224  46.4   238   270
Waiting:     111   224  46.4   238   270
Total:       114   227  46.6   241   274
#
Percentage of the requests served within a certain time (ms)
 50%    241
 66%    252
 75%    261
 80%    266
 90%    274
 95%    274
 98%    274
 99%    274
100%    274 (longest request)

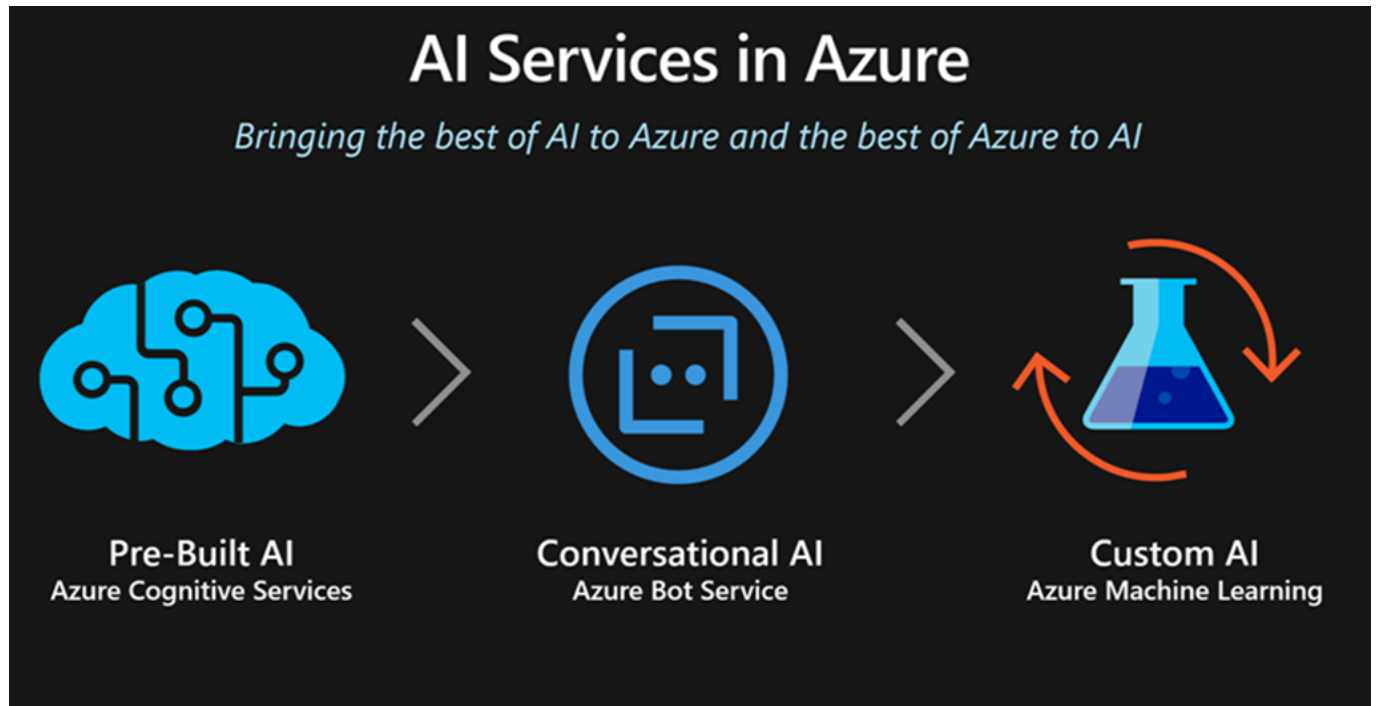
demouser@labvm MINGW64 ~/Desktop/nd00333_AZMLND_C2-master/starter_files
$ |
```

We can see that, calling 10 times the API with a `data.json` POST (the same used for testing the endpoint) we see that all of them are done without issues in 226 ms in average. This is a very acceptable responding time.

8. Create, Publish and Consume a Pipeline

1. Upload the Jupyter Notebook that contains the process of the Pipeline creation and change the experiment name so that it's the same used for AutoML.
2. Upload the `config.json` with the info about the workspace.
3. Run the AutoML step and publish the Pipeline so that we can find it in MLStudio Pipelines section.
4. Publish it using the recently created Pipeline Run (with run id).
5. Authenticate using interactive login and make a POST request with the new experiment name so that a new Pipeline run is executed.

Screen Recording



Standout Suggestions

For future work, I would suggest making further tweaks to the AutoML step of the pipeline. There are a lot of settings involved and making changes to them could improve the search space and help find an even better model solution. There are also additional steps that could be added to the pipeline, perhaps doing some dataset cleanup or feature engineering before the AutoML step, or doing additional steps after the AutoML step has completed.

In addition, I propose connecting Application Insights to our Pipeline API and Scheduling periodical runs or each time Banking Dataset updates (trigger).