SAVING THE TITANIC ONE ALGORITHM AT A TIME

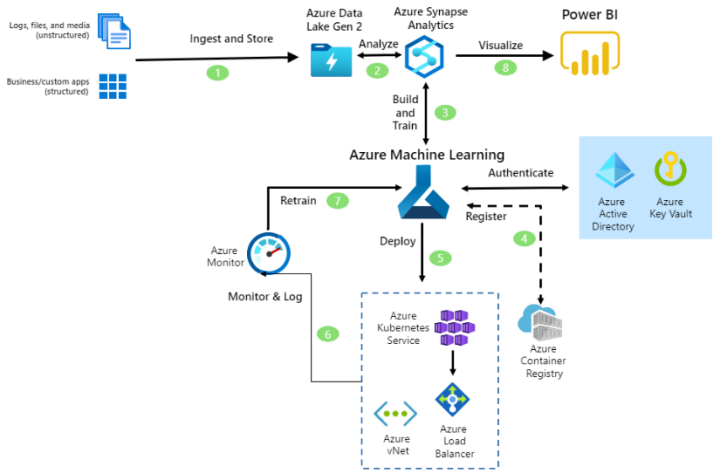
Implementing Automated ML using Python

-Azure Machine Learning Studio, Automated ML

PROBLEM STATEMENT:

State-of-the-art machine learning models and artificially intelligent systems consists of complex processes and deal with a lot of choices right from hyper parameter tuning to choosing models that provide a better accuracy and the metrics that govern this behavior. To achieve this result manually a lot of experiments consume a lot of time and compute resources are needed. There are many kinds of machine learning algorithms that you can use to train a model, and most of the time it becomes exceedingly difficult and complex to choose the most effective algorithm for your data and prediction requirements. Additionally, we perform a lot of preprocessing steps that help us prepare it for the ML models such as normalization, missing feature imputation, and others. In the quest to find the best model for our requirements, we play around with various permutations and combinations of algorithms and preprocessing transformations; which takes a lot of time and compute resources.

SOLUTION ARCHITECTURE:

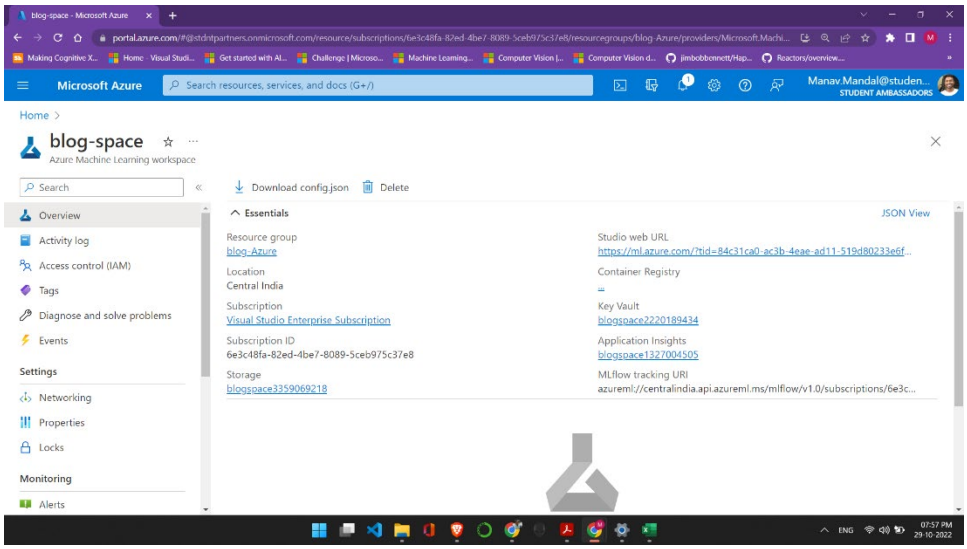


SOLUTION:

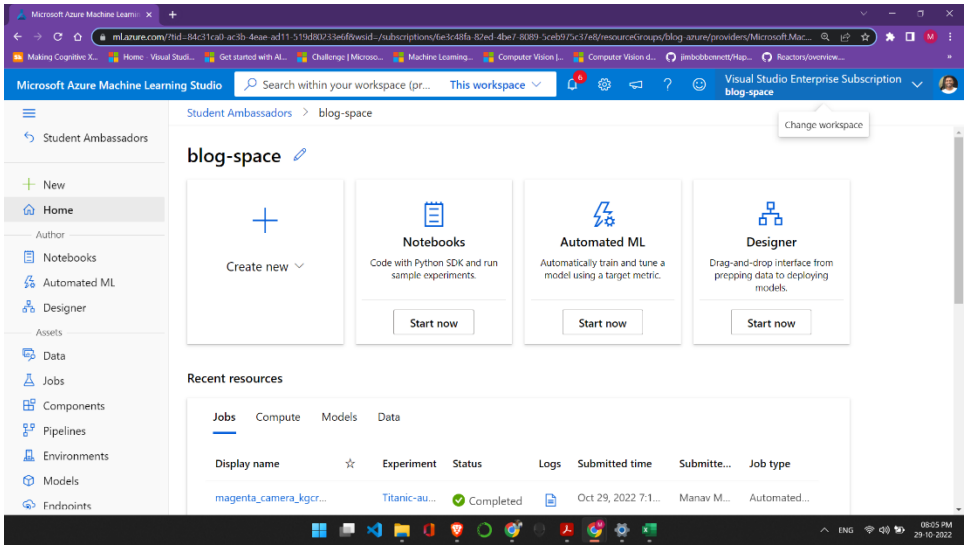
Now, I have spent a lot of time figuring out the perfect model for my project, tuning the hyper parameters so that I get better accuracy and trust me it gets exhausting. With the help of Azure Machine Learning we automate the comparison of models trained using different algorithms and preprocessing options. We can use the visual interface to interact with the studio online or we can use SDK's available to create personalized customizations. The only difference between these two methods is that SDK gives you greater control over the settings for the automated machine learning experiment, but the visual interface is easier to use.

We will explore the titanic dataset and understand how the entire process of machine learning gets automated. Before we continue let us look at what is Automated ML. This exercise will guide you how to you can use your Azure Subscription (https://azure.microsoft.com/en-us/free/) and Azure Machine Learning Studio to it automatically try multiple pre-processing techniques and model-training algorithms in parallel. Here we explore the power of cloud compute to find the best performing ML model for our data. Automated ML helps us to train models without an in-depth data science or programming knowledge. For people having some experience in data science and programming, it provides a way to save time and resources by efficiently automating the process of algorithm selection and hyperparameter tuning.

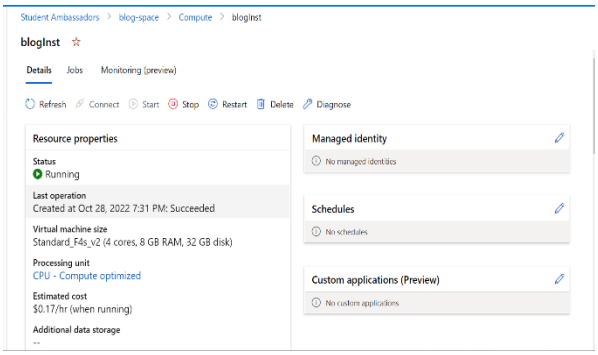
Let us start by creating a Machine Learning resource in our Azure cloud. I have named my resource blog-space but feel free to name it whatever you like and proceed with the default values.

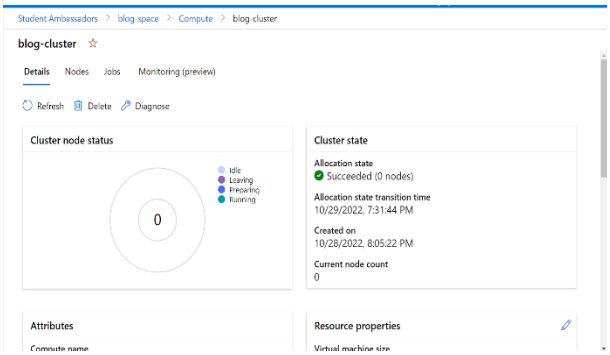


After creating the resource group, you should get a page like the one shown above, click on the Studio Web URL that will take you to the Machine Learning Studio.

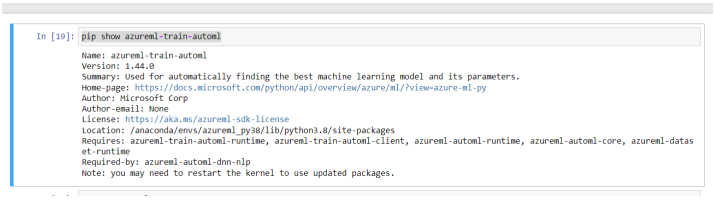


This is how the studio looks like and as we can see there are a lot of amazing features that can be and are being used by developers around the world. In the left column scroll down and click on compute. Here we will create our compute instance and compute cluster. We keep the default values, but you can select the VM according to your subscription. I have selected Standard_DS11_v2 but you are free to choose from the list.





In the Compute Instance click on the Jupyter option which opens the Jupyter Notebook (Make sure to not open Jupyter Lab). Next, we will create a new notebook and I have named my notebook Automated ML. Lets go through the notebook one cell at a time. In addition to the latest version of the **azureml-sdk** packages, we need the **azureml-train-automl** package to run the code in this notebook.



```
In [19]: pip show azureml-train-automl

Name: azureml-train-automl
Version: 1.44.0
Summary: Used for automatically finding the best machine learning model and its parameters.
Home-page: https://docs.microsoft.com/python/api/overview/azure/ml/?view=azure-ml-py
Author: Microsoft Corp
Author-email: None
License: https://aka.ms/azureml-sdk-license
Location: /anaconda/envs/azureml_py38/lib/python3.8/site-packages
Requires: azureml-train-automl-runtime, azureml-train-automl-client, azureml-automl-runtime, azureml-automl-core, azureml-dataset-runtime
Required-by: azureml-automl-dnn-nlp
Note: you may need to restart the kernel to use updated packages.
```

With the required SDK installed we are ready to connect to our workspace.

```
In [20]: import azureml.core
         from azureml.core import Workspace

         ws = Workspace.from_config()
         print("Ready to use Azure ML {} to work with {}".format(azureml.core.VERSION, ws.name))

         Ready to use Azure ML 1.44.0 to work with blog-space
```

We need to load the training data in our notebook. The code below looks complicated but all it is doing is searching the datastore for a dataset named Titanic. If it is not present, then upload the data and store it in the datastore.

```
In [21]: from azureml.core import Dataset

         default_ds = ws.get_default_datastore()

         if 'Titanic dataset' not in ws.datasets:
             default_ds.upload_files(files=['./Titanic.csv'], # Upload the Titanic csv file
                                     target_path='Titanic-data/', # Put it in a folder path in the datastore
                                     overwrite=True, # Replace existing files of the same name
                                     show_progress=True)

             #Create a tabular dataset from the path on the datastore
             tab_data_set = Dataset.Tabular.from_delimited_files(path=(default_ds, 'Titanic-data/*.csv'))

             # Register the tabular dataset
             try:
                 tab_data_set = tab_data_set.register(workspace=ws,
                                                      name='Titanic dataset',
                                                      description='Titanic data',
                                                      tags = {'format':'CSV'},
                                                      create_new_version=True)
                 print('Dataset registered.')
             except Exception as ex:
                 print(ex)
         else:
             print('Dataset already registered.')


         # Split the dataset into training and validation subsets
         diabetes_ds = ws.datasets.get("Titanic dataset")
         train_ds, test_ds = diabetes_ds.random_split(percentage=0.7, seed=123)
         print("Data ready!")

         Dataset already registered.
         Data ready!
```

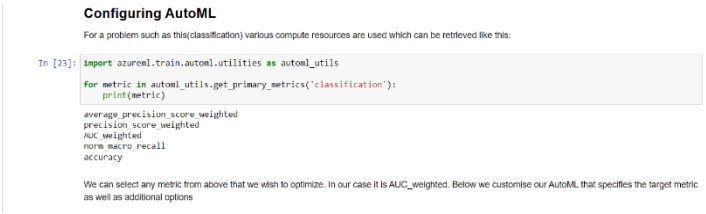
Remember the cluster we created before well let's connect to it here.

Using Compute Cluster

After creating a cluster in Azure machine learning studio we specify the compute cluster below

```
In [22]: from azureml.core.compute import ComputeTarget
         training_cluster = ComputeTarget(workspace=ws, name="blog-cluster")
```

One of the most important configuration settings is the metric by which model performance should be evaluated. You can retrieve a list of the metrics that are calculated for a particular type of model task like this:



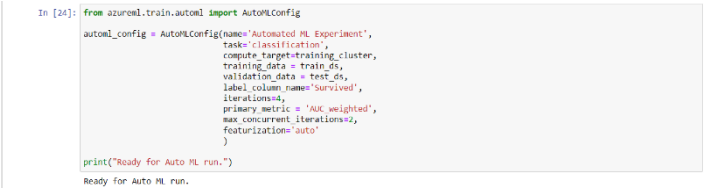
Configuring AutoML

For a problem such as this(classification) various compute resources are used which can be retrieved like this:

```
In [23]: import azureml.train.automl.utilities as automl_utils

         for metric in automl_utils.get_primary_metrics('classification'):
             print(metric)

average_precision_score_weighted
precision_score_weighted
AUC_weighted
norm_macro_recall
accuracy
```

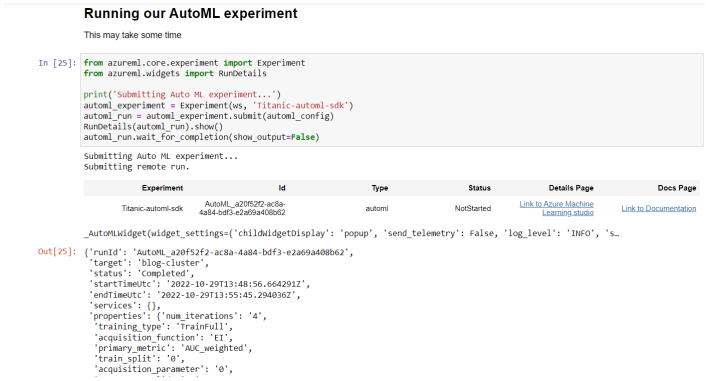
We can select any metric from above that we wish to optimize. In our case it is AUC_weighted. Below we customise our AutoML that specifies the target metric as well as additional options

Having decided the metric, you want to optimize (*AUC_weighted*), you can configure the automated machine learning run. Since this is a simple dataset I have kept the number of iterations to 4.



```
In [24]: from azureml.train.automl import AutoMLConfig

         automl_config = AutoMLConfig(name='Automated ML Experiment',
                                      task='classification',
                                      compute_target=training_cluster,
                                      training_data = train_ds,
                                      validation_data = test_ds,
                                      label_column_name='Survived',
                                      iterations=4,
                                      primary_metric = 'AUC_weighted',
                                      max_concurrent_iterations=2,
                                      featurization='auto'
                                      )

         print("Ready for Auto ML run.")

Ready for Auto ML run.
```

With all configurations set we are now ready to run our experiment. I have set *show_output = False* but you can set it to True to see the model being run in real-time.



Running our AutoML experiment

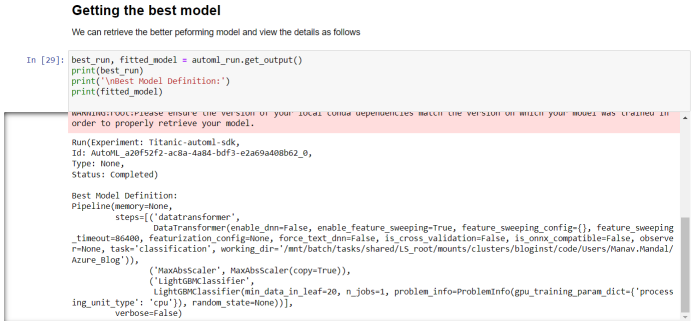
This may take some time

```
In [25]: from azureml.core.experiment import Experiment
         from azureml.widgets import RunDetails

         print('Submitting Auto ML experiment...')
         automl_experiment = Experiment(ws, 'Titanic-automl-sdk')
         automl_run = automl_experiment.submit(automl_config)
         RunDetails(automl_run).show()
         automl_run.wait_for_completion(show_output=False)

Submitting Auto ML experiment...
Submitting remote run.
```

Experiment	Id	Type	Status	Details Page	Docs Page
Titanic-automl-sdk	AutoML_a20f52f2-ac8a-4a84-bdf3-e2a69a408b62	automl	NotStarted	Link to Azure Machine Learning studio	Link to Documentation

```
_AutoMLWidget(widget_settings={'childWidgetDisplay': 'popup', 'send_telemetry': False, 'log_level': 'INFO', 's…

Out[25]: {'runId': 'AutoML_a20f52f2-ac8a-4a84-bdf3-e2a69a408b62',
          'target': 'blog-cluster',
          'status': 'Completed',
          'startTimeUtc': '2022-10-29T13:48:56.664291Z',
          'endTimeUtc': '2022-10-29T13:55:45.294036Z',
          'services': {},
          'properties': {'num_iterations': '4',
           'training_type': 'TrainFull',
           'acquisition_function': 'EI',
           'primary_metric': 'AUC_weighted',
           'train_split': '0',
           'acquisition_parameter': '0',
```

We can retrieve the best performing as done below and additionally we can view the best run transformation and best run metrics as well.



Getting the best model

We can retrieve the better peforming model and view the details as follows

```
In [29]: best_run, fitted_model = automl_run.get_output()
         print(best_run)
         print('\nBest Model Definition:')
         print(fitted_model)
```

```
WARNING:root:Please ensure the version of your local conda dependencies match the version on which your model was trained in
order to properly retrieve your model.

Run(Experiment: Titanic-automl-sdk,
Id: AutoML_a20f52f2-ac8a-4a84-bdf3-e2a69a408b62_0,
Type: None,
Status: Completed)

Best Model Definition:
Pipeline(memory=None,
         steps=[('datatransformer',
                 DataTransformer(enable_dnn=False, enable_feature_sweeping=True, feature_sweeping_config={}, feature_sweeping
_timeout=86400, featurization_config=None, force_text_dnn=False, is_cross_validation=False, is_onnx_compatible=False, observe
r=None, task='classification', working_dir='/mnt/batch/tasks/shared/LS_root/mounts/clusters/bloginst/code/Users/Manav.Mandal/
Azure_Blog')),
                ('MaxAbsScaler', MaxAbsScaler(copy=True)),
                ('LightGBMClassifier',
                 LightGBMClassifier(min_data_in_leaf=20, n_jobs=1, problem_info=ProblemInfo(gpu_training_param_dict={('process
ing_unit_type': 'cpu')}), random_state=None))],
         verbose=False)
```

Finally, having found the best performing model, you can register it.

Saving the Model

Finally we can register the best fitting model

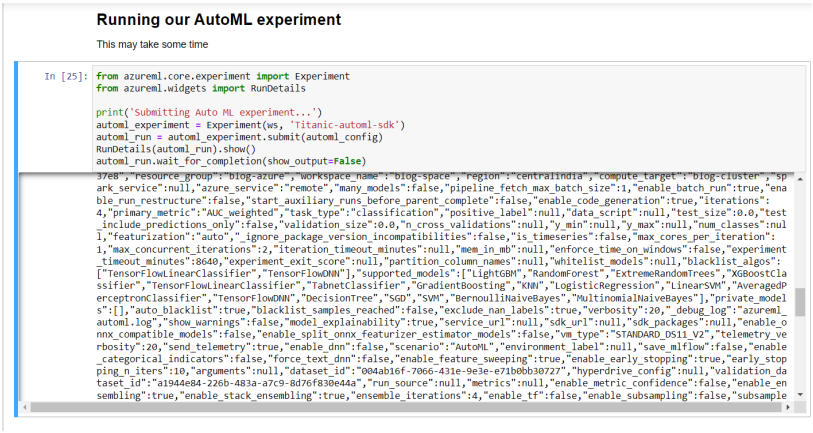
```
In [28]: from azureml.core import Model

         # Register model
         best_run.register_model(model_path='outputs/model.pkl', model_name='Titanic_model',
                                 tags={'Training context':'Auto ML'},
                                 properties={'AUC': best_run_metrics['AUC_weighted'], 'Accuracy': best_run_metrics['accuracy']})

         # List registered models
         for model in Model.list(ws):
             print(model.name, 'version:', model.version)
             for tag_name in model.tags:
                 tag = model.tags[tag_name]
                 print ('\t',tag_name, ':', tag)
             for prop_name in model.properties:
                 prop = model.properties[prop_name]
                 print ('\t',prop_name, ':', prop)
             print('\n')
```

```
Titanic_model version: 4
        Training context : Auto ML
        AUC : 0.8465673385918785
        Accuracy : 0.8171641791044776
```

CHALLENGES FACED:

Initially when trying out Automated ML in my local system I faced a few issues regarding connecting my Azure subscription with Visual Studio Code. I couldn't find a solution for it and hence moved on to Azure Cloud. It turned out to be perfect as it eliminated the need to create a JSON file which required the endpoints and subscription key.

The outputs of some of the code cells were a little difficult to understand. For someone having little to no knowledge about this domain the output may look like gibberish. For example, when we submit the AutoML experiment we get an extensive list of values including not only the models used but also the dependencies and their versions. I had to spend some time figuring out and understanding the output.



BUSINESS BENEFITS:

All Azure Machine Learning does is help data scientists and developers to build, deploy, and manage high-quality models faster and with 100% confidence. With such large and complex operations in place industries need a solution that can be brought into production as soon as possible and reliable at the same time. Some features are as follows

1. Open-source interoperability
2. rapid model training and deployment
3. integrated tools

This tool helps a development experience that supports the entire machine learning process right from building to training and deploying models. Different models require different formats of input data, we eliminate this problem by developing accurate models with automated machine learning for image, textual or even tabular models by using feature engineering and tuning the hyperparameters. We can even use Visual Studio Code to go from local to cloud training smoothly and autoscale with powerful cloud-based CPU and GPU clusters.

We can evaluate machine learning models with reproducible and automated workflows to verify the

1. explainability,
2. error analysis,
3. causal analysis,
4. model performance,
5. exploratory data analysis,

Contextualize responsible AI metrics for both technical and non-technical audiences to involve stakeholders and streamline compliance review. So that's it from my side. Happy Coding!

-Manav Mandal (Student at MESCOE)

https://github.com/MXNXV/Automated-ML

https://www.linkedin.com/in/manav-mandal-5b1496196/