Assignment 4.1 – SIT 788

Target grade: High Distinction

Part 1: Azure ML Studio

Model Training

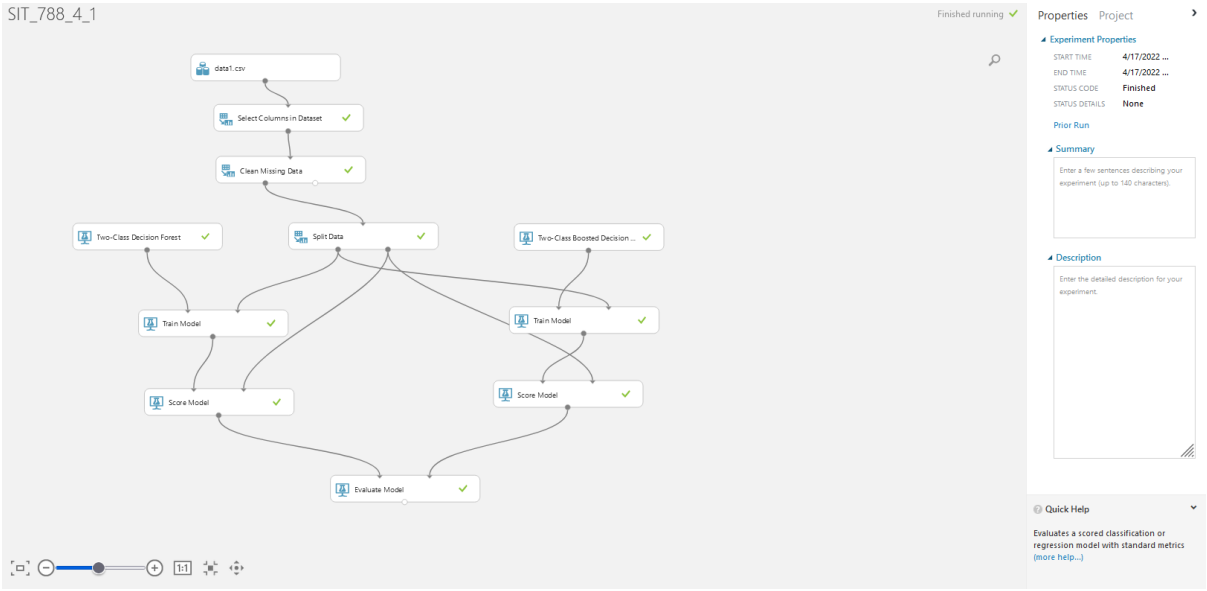

Fig 1: Trained models: left branch – Random Forest, right branch – Decision Tree

Model comparison

Decision Tree

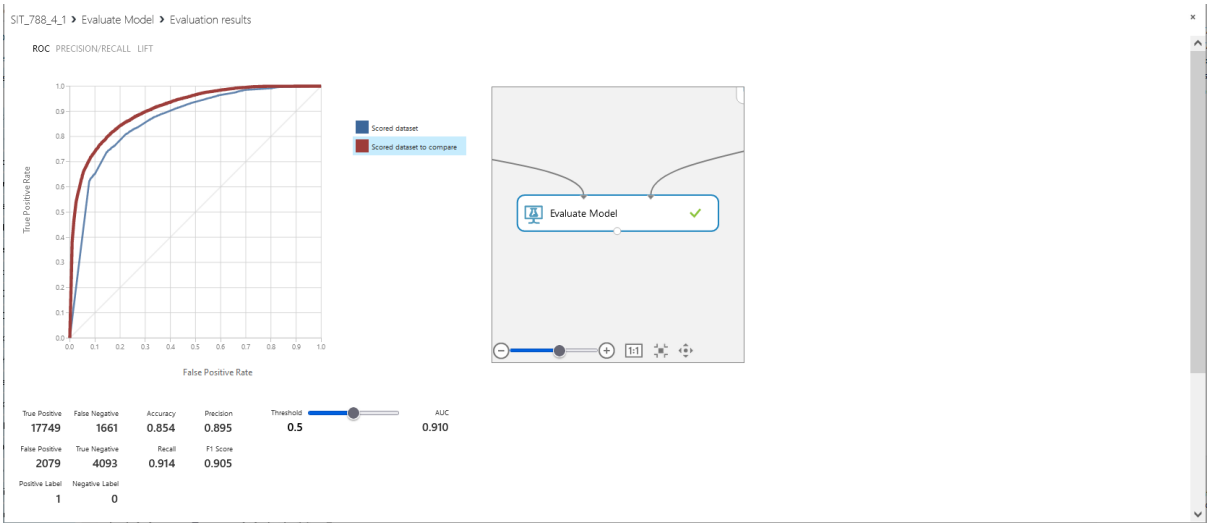

Fig 2: Evaluation Metrics: Decision Tree

Random Forest

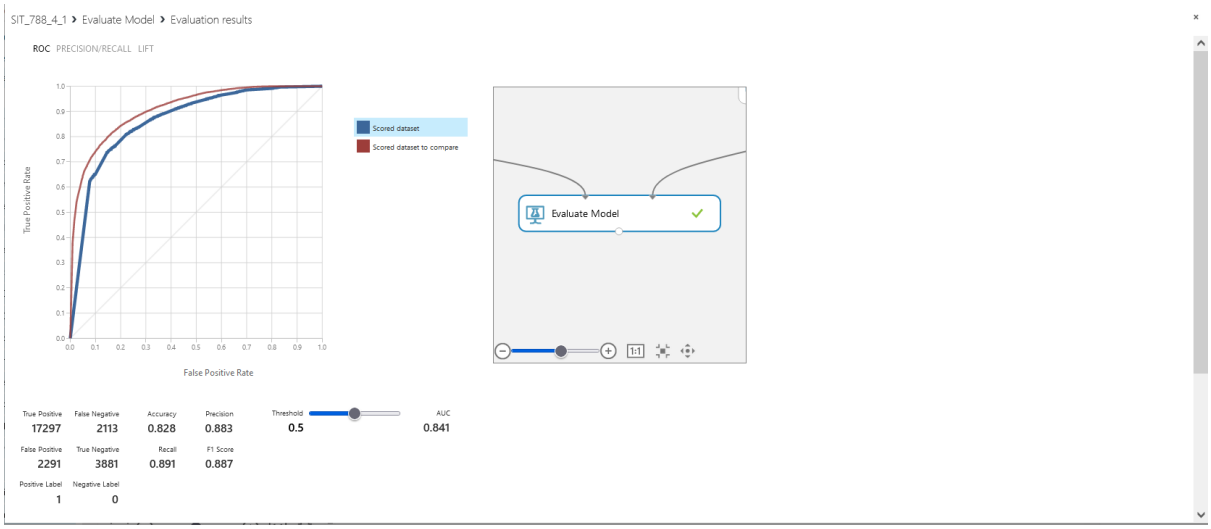

Fig 3: Evaluation Metrics: Random Forest

From the above figures, fig 2 and fig 3, it is evident that for the given set of parameters Decision Tree outperforms Random Forest on this data (data1.csv from week 2 seminar):

Accuracy of Decision Tree: 85.4%, **Precision**: 89.5%, **Recall** 91.4%

Accuracy of Random Forest: 82.8%, **Precision**: 88.3%, **Recall** 89.1%

This shows that it is not always necessary that ensemble models outperform their constituent models. Ensemble models are better when single model have problems converging because of the complexities of the dataset, here the dataset is simple which leads to Decision Trees getting higher evaluation metrics across all categories.

Note: It is worth highlighting that for the purpose of this experiment default parameters were used, the results and performance of both the classifiers can change if hyperparameter optimization is done.

Deployed model

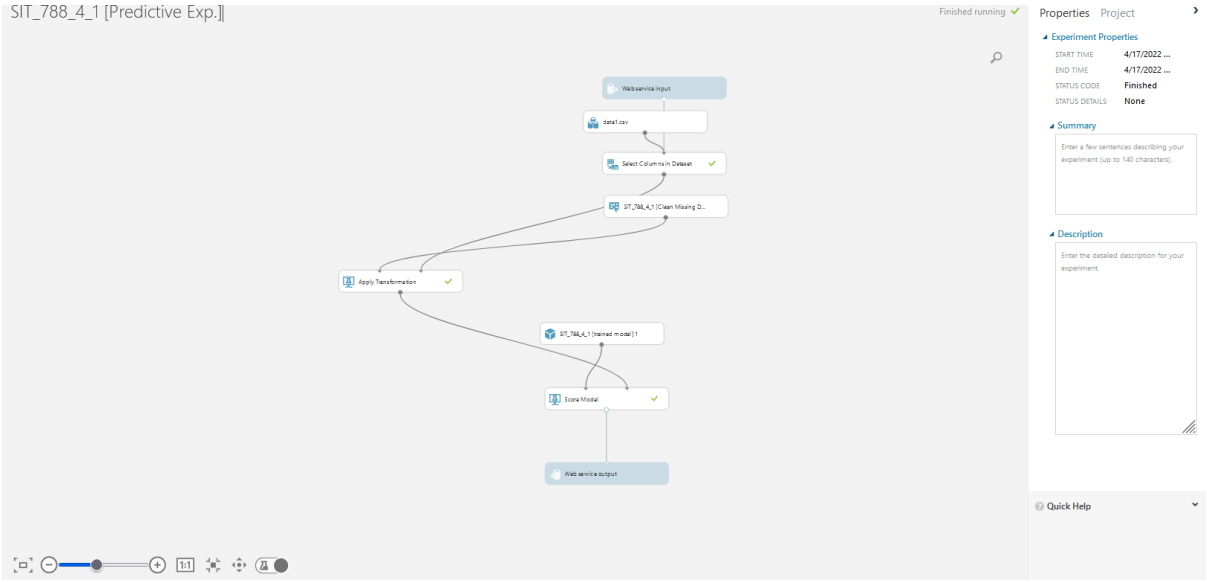

Fig 4: Deployed best model (Decision Tree)

Following figures denote the API key of the deployed model and test run using the deployed web service for prediction on sample data.

Fig 5: Deployed Model API Key

Fig 6: Figure denoting deployed model test result.

Part 2: Azure Machine Learning Python SDK

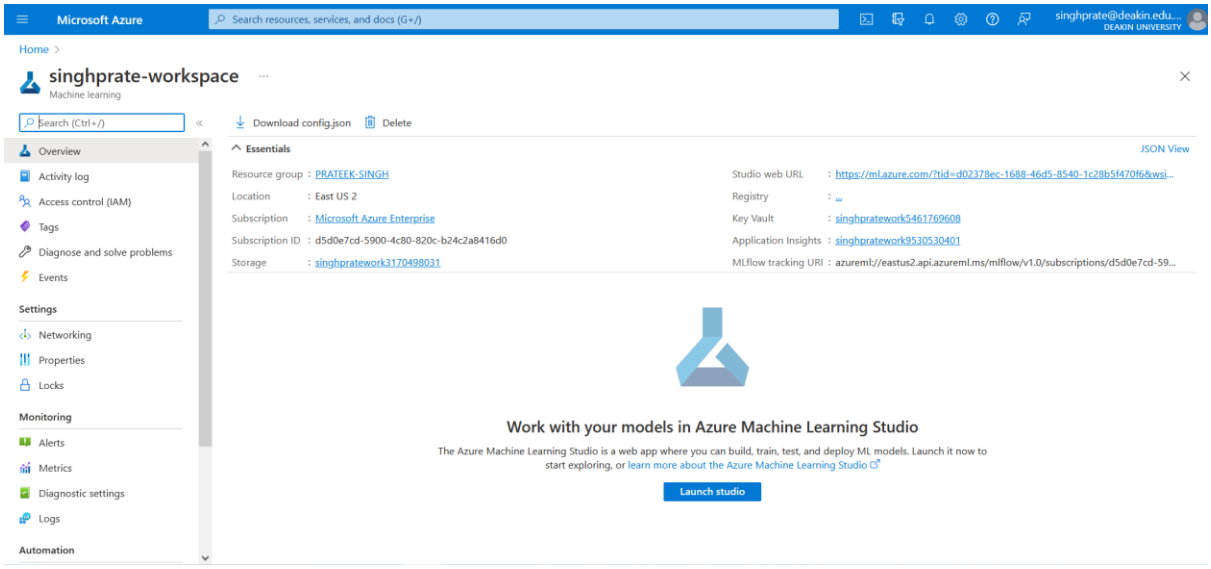

Fig 7: Overview of the built workspace.

As in the previous step we have seen that Decision Tree model is the best trained model therefore we train only Decision Tree on “data1.csv” from week 2 practical.

In [1]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
from time import time
```

In [2]:
```
#### Load Data
df = pd.read_csv("SIT788_4_1_Data/data1.csv")
```

In [3]:
```
print(df.isnull().sum())
display(df)
```

```
f1       0
f2       0
f3       0
f4       0
f5       0
f6       0
f7       0
f8       0
f9       0
f10      0
f11      0
f12      0
f13      0
f14      0
class    0
dtype: int64
```

Fig 8: Imports, loading data and null check.

As the data contains categorical string variables, therefore we perform label encoding.

Handle categorical variables

In [4]:
```
labelEncoder = preprocessing.LabelEncoder()
df = df.apply(labelEncoder.fit_transform)
display(df)
```

	f1	f2	f3	f4	f5	f6	f7	f8	f9	f10	f11	f12	f13	f14	class
0	22	7	2671	9	12	4	1	1	4	1	25	0	39	39	1
1	33	6	2926	9	12	2	4	0	4	1	0	0	12	39	1
2	21	4	14086	11	8	0	6	1	4	1	0	0	39	39	1
3	36	4	15336	1	6	2	6	0	2	1	0	0	39	39	1
4	11	4	19355	9	12	2	10	5	2	0	0	0	39	5	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32556	10	4	16528	7	11	2	13	5	4	0	0	0	37	39	1
32557	23	4	8080	11	8	2	7	0	4	1	0	0	39	39	0
32558	41	4	7883	11	8	6	1	4	4	0	0	0	39	39	1
32559	5	4	12881	11	8	4	1	3	4	1	0	0	19	39	1
32560	35	5	17825	11	8	2	4	5	4	0	108	0	39	39	0

32561 rows × 15 columns

Fig 9: Label Encoding of string data

Split into train and test ¶

In [5]:
```
X, Y = df.loc[:, (df.columns != 'class')], df[['class']]
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.20, random_state=42)
```

Train best model (decision tree)

In [6]:
```
clf_dt = DecisionTreeClassifier()

millis_a = int(time() * 1000)
clf_dt.fit(x_train, y_train)
millis_b = int(time() * 1000)
dif = millis_b - millis_a
print ("Decsion Tree train time: ", dif)

millis_a = int(time() * 1000)
y_pred_dt = clf_dt.predict(x_test)
millis_b = int(time() * 1000)
dif = millis_b - millis_a
avg_t = dif / len(x_test)
print ("Decsion Tree time per prediction: ", avg_t)
```

```
Decsion Tree train time:  178
Decsion Tree time per prediction:  0.0009212344541685859
```

In [7]:
```
print(accuracy_score(y_test, y_pred_dt))
print(confusion_matrix(y_test, y_pred_dt))
```

```
0.812375249500998
[[ 992  579]
 [ 643 4299]]
```

Fig 10: Splitting data, training and validation of the trained model.

Deploy to Azure Workspace

Install deps

In [8]:
```
#!pip install azureml-core
#!pip install --upgrade azureml-core
```

Import Azure ML python SDK

In [14]:
```
import azureml.core
print(azureml.core.VERSION)
from azureml.core import Workspace
from azureml.core.model import Model
```

```
1.40.0
```

Connect to created Workspace

In [10]:
```
ws = Workspace.get(name="singhprate-workspace", subscription_id='d5d0e7cd-5900-4c80-820c-b24c2a8416d0',
                   resource_group='PRATEEK-SINGH')
```

Fig 11: Install dependencies and connect to the created workspace in fig 7

After successful connection to the workspace, we dump our model to disk and then register our model:

Dump trained model

In [12]:
```
joblib.dump(clf_dt, "SIT788_4_1_Data/decision_tree_v1.pkl")
```

Out[12]:
```
['SIT788_4_1_Data/decision_tree_v1.pkl']
```

Register model on Workspace

In [24]:
```
model = Model.register(model_path="SIT788_4_1_Data/decision_tree_v1.pkl",
                       model_name="decision_tree_classification_model",
                       model_framework=Model.Framework.SCIKITLEARN,
                       model_framework_version=sklearn.__version__,
                       tags={'type': "classification"},
                       description="Decsion Tree binary classification model",
                       workspace=ws)
```

```
Registering model decision_tree_classification_model
```

Fig 12: Dump the model and register the model using Azure ML python SDK

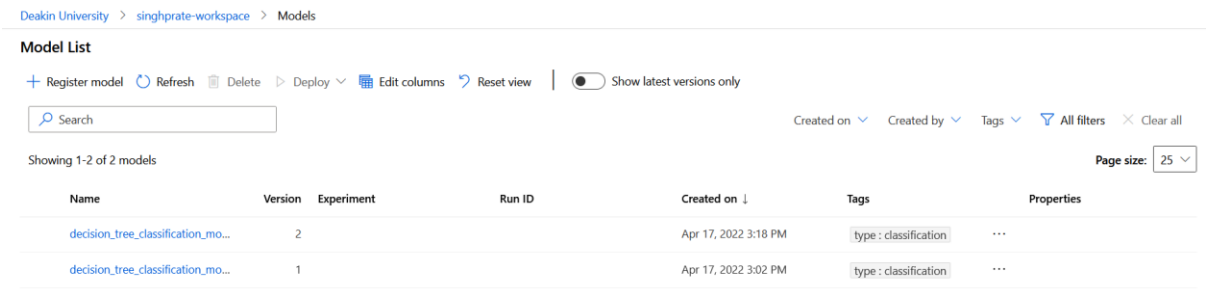

Fig 13: After registering the model, we can see all the registered models in our workspace.

Deployment step

Finally, after the model has been registered, we deploy it as a webservice and test it using dummy data. [This step takes time]. After the service has been used, we delete it to free the memory.

Deploy model to workspace

```
In [25]: service_name = 'decision-tree-service'
         service = Model.deploy(ws, service_name, [model], overwrite=True)
         service.wait_for_deployment(show_output=True)
```

```
Tips: You can try get_logs(): https://aka.ms/debugimage#dockerlog or local deployment: https://aka.ms/debugimage#debug-locally to debug if deployment takes longer than 10 minutes.
Running
2022-04-17 15:18:32+05:30 Creating Container Registry if not exists..
2022-04-17 15:28:32+05:30 Registering the environment.
2022-04-17 15:28:33+05:30 Uploading autogenerated assets for no-code-deployment.
2022-04-17 15:28:34+05:30 Building image..
2022-04-17 15:32:30+05:30 Generating deployment configuration.
2022-04-17 15:32:34+05:30 Submitting deployment to compute..
2022-04-17 15:32:49+05:30 Checking the status of deployment decision-tree-service..
2022-04-17 15:33:42+05:30 Checking the status of deployment decision-tree-service..
2022-04-17 15:34:42+05:30 Checking the status of inference endpoint decision-tree-service.
2022-04-17 15:34:44+05:30 Checking the status of inference endpoint decision-tree-service.
Succeeded
ACI service creation operation finished, operation "Succeeded"
```

Test on dummy data

```
In [31]: import json

         input_payload = json.dumps({
             'data': x_test[0:2].values.tolist(),
             'method': 'predict'  # If you have a classification model, you can get probabilities by changing this to 'predict_proba
         })

         output = service.run(input_payload)

         print(output)
```

```
{'predict': [1, 1]}
```

Delete service after use

```
In [ ]: service.delete()
```

Fig 14: Deploying the model as webservice and testing the service with dummy data.

We can also check the endpoints at the Azure ML workspace to see the status of the deployed webservice.