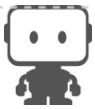


How to operationalize DataRobot API

With MS SQL Server 2017 and Python

Before we go through this tutorial, let's first take a look at

Why you should care about operationalizing your model?

Traditionally, implementing new models has been a very time consuming two-stage process. During the first stage, model features are identified, a model is fitted, trained, and then built (for more information click here). This is typically an infrequent process, which uses historical data, and depending on the model and underlying data, it can take months to complete.
In the second stage, the best performing model is deployed and integrated into a production application, to generate predictions. For a typical enterprise, this deployment can take anywhere between 12 to 20 weeks to complete and the cost to deploy one model can run in excess of $250,000. Therefore, it is no surprise that only less than 10% of models make it into production.
One of the reasons for this long deployment window and cost overruns is that Data Scientists typically write models in R/Python, and developers have to rewrite the model in other languages (e.g. Java or C#). Additionally, app developers have to add more code to integrate the model itself.
Only if a model is used in a production application, a model actually generates value to the business. Yet, due to data drift, it might only be several weeks until the model has to be re-trained, and the process starts all over again in stage 1. If the model is not actively monitored or no corrective measure is taken, a business runs the risk of making incorrect decisions, and the ROI of a given model is dwindling.

It is essential for a model to not only be easy to train and guarantee great performance, but also that it is easy to integrate, and can be updated in an automated fashion. This is where the DataRobot AI API comes into play. The DataRobot AI API allows you to automate the model training process without requiring any Data Science knowledge. In addition, it connects the first stage with the second stage, allowing a true end to end process automation from model training to deployment in a production application.

In this tutorial, I will demonstrate how to use the DataRobot API to train a model directly from a table within MS SQL Server (stage 1), and subsequently how to use the DataRobot Prediction API with data from a table in MS SQL Server. This integration pattern is commonly referred to as "In-database Scoring". While there are many other integration patterns, the advantage of this approach is that it is fairly simple, due to the small number of components, as well as very cost effective, because it does not require any dedicated hardware or software.

Since the release of MS SQL Server 2016 with R Services and MS SQL Server 2017 with ML Services, it has become a lot easier to re-use the data preparation and enrichment work that has been done in R or Python in an actual production pipeline. Not only does this save a lot of time and money, but it also ensures consistency between stage 1 and stage2, which is often overlooked and a common pitfall in the process of "productionalizing" a model. For this reason

alone, re-using an existing R or Python script is favorable over rewriting the logic in a CLR procedure in C# (even though that is still possible).

What is the DataRobot API and how can you use it?

The DataRobot API automates the process of creating highly accurate AI-powered applications. In contrast to it's free counterpart, the DataRobot AI API, it supports more powerful ML algorithms (such as Time Series) and monitoring options. Because it is a REST API, with only a few lines of code, you can get immediate access to 100's of different machine learning models. Also you can use the programming language that is most convenient and familiar to you, as you are not confined to just one programming language, due to the API or underlying ML algorithms.

Now that we know what the DataRobot API is all about, let's take a look at the use case and how you can use the API from Python.

What are we predicting?

For this tutorial we will utilize the synthetic 10k_diabetes dataset.

This dataset contains hospital admission data, which we would like to use to predict whether or not a patient has to be readmitted to the hospital within the following 30 days. This is a very common use case in the healthcare industry, because hospitals lose millions of dollars on readmission penalties and additional treatments each year.

Did You Know?
Hospital readmissions that occur within 30 days of discharge cost Medicare approximately $17 billion.

Also patient satisfaction takes a hit with every readmission, as an added complication of improper care. Therefore it is crucial for hospitals to build an accurate predictive model for readmissions and to incorporate it into their patient discharge considerations.

How can you use the DataRobot API?

First you will have to create an account on https://app.datarobot.com or install the DataRobot platform onPremise, and retrieve the API_TOKEN for your user.

Once this is done, it is actually quite simple to use the DataRobot API.
In this tutorial we will call the API from Python, thus make use of it's dedicated python client.
You can install the DataRobot API client with the following command:

```
pip install datarobot
```

Once you have installed the client, you can use it in your python scripts. As with any other library, the first step is to reference the library in your script.

```
import datarobot as dr
```

Then, to actually use the API via the client, you will have to authenticate (see below).

```
drclient = dr.Client(endpoint=APIENDPOINT, token=API_TOKEN)
```

Once you have done this, you can create a project and kick off the model training as shown below.

```
project = dr.Project.start(sourcedata=TrainingDataSet,
target='is_bad', autopilot_on=True)
```

To generate predictions, a dedicated prediction server is used to ensure a reliable scoring environment.

```
predictions_response =
requests.post("%s/deployments/%s/predictions" %
(PREDICTIONSAPIENDPOINT,DEPLOYMENT_ID),auth=(USERNAME,
API_TOKEN), data=PredictionsDataSet, headers=HEADERS)
```

Now let's take a look at how we can use the above commands from MS SQL server in a production pipeline.

How to create a new model within MS SQL Server

You can download the SQL script, which contains all the required SQL tables, procedures and triggers here.

Before we can use a Python or R script from within MS SQL server, we have to ensure that ML Services is installed. To verify that ML services is actually available, you can run the below T-SQL.

```
EXECUTE sp_execute_external_script
@language =N'Python',
@script=N'import sys
print(sys.version)';
```

If it is not installed yet, you can follow the below guide for the installation of ML services*.
https://docs.microsoft.com/en-us/sql/advanced-analytics/install/sql-machine-learning-services-windows-install?view=sql-server-2017

After we verified that ML services is installed, we have to enable the execution of external scripts by using the below T-SQL.

```
sp_configure 'external scripts enabled', 1;
```

Then we have to reboot the SQL server instance so that the configuration change becomes effective.
The last step is to install the DataRobotAI library. To do this, we have to open the command prompt and navigate to the Python directory of MS SQL server:

`C:\Program Files\Microsoft SQL Server\MSSQL14.MSSQLSERVER\PYTHON_SERVICES`

If the above path is not the correct directory, you can use the below T-SQL statement to identify your path:

```
EXEC sp_execute_external_script
  @language =N'Python',
  @script=N'import sys; print("\n".join(sys.path))'
```

Within the Python directory, you can use pip to install all dependencies as usual. For this tutorial we will install the following library with the below command:

```
pip install datarobot
```

** Please note, I had to install the latest SQL server update, otherwise the function "sp_execute_external_script" was not executed properly.*

Finally, we create a table called “admissions” and import the sample dataset. The only modification we have to make to the columns in the csv file is that we have to add an id column, remove spaces in the column names, and add a few more columns to the table to store the prediction results (highlighted in bold).

```
CREATE TABLE [dbo].[admissions](
        [id] int NOT NULL IDENTITY,
        [race] [nvarchar](4000) NULL,
        [gender] [nvarchar](4000) NULL,
        [age] [nvarchar](4000) NULL,
        [weight] [nvarchar](4000) NULL,
        [admission_type_id] [nvarchar](4000) NULL,
        [discharge_disposition_id] [nvarchar](4000) NULL,
        [admission_source_id] [nvarchar](4000) NULL,
        [time_in_hospital] [nvarchar](4000) NULL,
        [payer_code] [nvarchar](4000) NULL,
        [medical_specialty] [nvarchar](4000) NULL,
        [num_lab_procedures] [nvarchar](4000) NULL,
        [num_procedures] [nvarchar](4000) NULL,
        [num_medications] [nvarchar](4000) NULL,
        [number_outpatient] [nvarchar](4000) NULL,
        [number_emergency] [nvarchar](4000) NULL,
        [number_inpatient] [nvarchar](4000) NULL,
        [diag_1] [nvarchar](4000) NULL,
        [diag_2] [nvarchar](4000) NULL,
        [diag_3] [nvarchar](4000) NULL,
        [number_diagnoses] [nvarchar](4000) NULL,
        [max_glu_serum] [nvarchar](4000) NULL,
        [A1Cresult] [nvarchar](4000) NULL,
        [metformin] [nvarchar](4000) NULL,
        [repaglinide] [nvarchar](4000) NULL,
        [nateglinide] [nvarchar](4000) NULL,
        [chlorpropamide] [nvarchar](4000) NULL,
        [glimepiride] [nvarchar](4000) NULL,
        [acetohexamide] [nvarchar](4000) NULL,
        [glipizide] [nvarchar](4000) NULL,
        [glyburide] [nvarchar](4000) NULL,
        [tolbutamide] [nvarchar](4000) NULL,
        [pioglitazone] [nvarchar](4000) NULL,
        [rosiglitazone] [nvarchar](4000) NULL,
        [acarbose] [nvarchar](4000) NULL,
        [miglitol] [nvarchar](4000) NULL,
        [troglitazone] [nvarchar](4000) NULL,
        [tolazamide] [nvarchar](4000) NULL,
        [examide] [nvarchar](4000) NULL,
        [citoglipton] [nvarchar](4000) NULL,
        [insulin] [nvarchar](4000) NULL,
        [glyburidemetformin] [nvarchar](4000) NULL,
        [glipizidemetformin] [nvarchar](4000) NULL,
        [glimepiridepioglitazone] [nvarchar](4000) NULL,
        [metforminrosiglitazone] [nvarchar](4000) NULL,
        [metforminpioglitazone] [nvarchar](4000) NULL,
        [change] [nvarchar](4000) NULL,
        [diabetesMed] [nvarchar](4000) NULL,
        [readmitted] [nvarchar](4000) NULL,
        [diag_1_desc] [nvarchar](4000) NULL,
        [diag_2_desc] [nvarchar](4000) NULL,
        [diag_3_desc] [nvarchar](4000) NULL,
        [predictionValue] [nvarchar](500) NULL,
        [predictionLabel] [nvarchar](500) NULL,
        [prediction] [nvarchar](500) NULL,
        [predictionDate] [nvarchar](500) NULL,
        [predictionThreshold] [nvarchar](500) NULL
)
```

Lastly, we create a configuration table to store the API token, username, deployment_id, and target.

```
CREATE TABLE [dbo].[DRconfiguration](
	[API_KEY] [nvarchar](max) NULL,
	[USERNAME] [nvarchar](max) NULL,
	[PROJECTNAME] [nvarchar](max) NULL,
	[MODELMANAGEMENTENDPOINT] [nvarchar](max) NULL,
	[PREDICTIONSENDPOINT] [nvarchar](max) NULL,
	[DR_KEY] [nvarchar](max) NULL,
	[INSTANCE_ID] [nvarchar](max) NULL,
	[DEPLOYMENT_ID] [nvarchar](max) NULL,
	[WORKERCOUNT] [int] NULL,
	[TARGET] [nvarchar](max) NULL
)
```

Now we are ready to create and execute our python script from within MS SQL server.

How to make predictions within MS SQL Server

In order to train models directly from within MS SQL server and to create predictions, we first have to create a stored procedure, which includes our Python script. Then we need to create a trigger which invokes the procedure every time a new admission is added to the admissions table.

Create SQL stored procedure

The first step is to create a stored procedure, which encapsulates the Python script. To create the procedure, use the below code.

```
CREATE PROC [dbo].[sp_predictAdmissions] (@RETRAIN int = 1)
AS
BEGIN
	SET NOCOUNT ON
	DECLARE @PREDICTIONS TABLE(
	id int,
	race nvarchar(4000) ,
	gender nvarchar(4000) ,
	age nvarchar(4000) ,
	weight nvarchar(4000) ,
	admission_type_id nvarchar(4000) ,
	discharge_disposition_id nvarchar(4000) ,
	admission_source_id nvarchar(4000) ,
	time_in_hospital nvarchar(4000) ,
	payer_code nvarchar(4000) ,
	medical_specialty nvarchar(4000) ,
	num_lab_procedures nvarchar(4000) ,
	num_procedures nvarchar(4000) ,
```

```
            num_medications nvarchar(4000) ,
            number_outpatient nvarchar(4000) ,
            number_emergency nvarchar(4000) ,
            number_inpatient nvarchar(4000) ,
            diag_1 nvarchar(4000) ,
            diag_2 nvarchar(4000) ,
            diag_3 nvarchar(4000) ,
            number_diagnoses nvarchar(4000) ,
            max_glu_serum nvarchar(4000) ,
            A1Cresult nvarchar(4000) ,
            metformin nvarchar(4000) ,
            repaglinide nvarchar(4000) ,
            nateglinide nvarchar(4000) ,
            chlorpropamide nvarchar(4000) ,
            glimepiride nvarchar(4000) ,
            acetohexamide nvarchar(4000) ,
            glipizide nvarchar(4000) ,
            glyburide nvarchar(4000) ,
            tolbutamide nvarchar(4000) ,
            pioglitazone nvarchar(4000) ,
            rosiglitazone nvarchar(4000) ,
            acarbose nvarchar(4000) ,
            miglitol nvarchar(4000) ,
            troglitazone nvarchar(4000) ,
            tolazamide nvarchar(4000) ,
            examide nvarchar(4000) ,
            citoglipton nvarchar(4000) ,
            insulin nvarchar(4000) ,
            glyburidemetformin nvarchar(4000),
            glipizidemetformin nvarchar(4000),
            glimepiridepioglitazone nvarchar(4000) ,
            metforminrosiglitazone nvarchar(4000) ,
            metforminpioglitazone nvarchar(4000) ,
            change nvarchar(4000) ,
            diabetesMed nvarchar(4000) ,
            readmitted nvarchar(4000) ,
            diag_1_desc nvarchar(4000) ,
            diag_2_desc nvarchar(4000) ,
            diag_3_desc nvarchar(4000) ,
            prediction nvarchar(500),
            predictionThreshold nvarchar(max),
            predictionLabel nvarchar(500),
            predictionValue nvarchar(500),
            DEPLOYMENT_ID nvarchar(500),
            predictionDate nvarchar(500)
)

DECLARE @PSCRIPT NVARCHAR(MAX);
SET @PSCRIPT = N'
import pandas as pd
import datarobot as dr
from datetime import date
import time
from pandas import DataFrame
import urllib3
import requests
urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)
import json

def wait_for_async_resolution(client, status_url):
   status = False
   while status == False:
      resp = client.get(status_url)
      try:
         r = json.loads(resp.content.decode("utf-8"))
```

```
            if resp.status_code == 200 and r["status"].upper() == "ACTIVE":
                status = True
                return resp
        except:
            print("error")
        time.sleep(10)   # Delays for 10 seconds.

def wait_for_result(client, response):
    assert response.status_code in (200, 201, 202), response.content

    if response.status_code == 200:
        data = response.json()

    elif response.status_code == 201:
        status_url = response.headers["Location"]
        resp = client.get(status_url)
        assert resp.status_code == 200, resp.content
        data = resp.json()

    elif response.status_code == 202:
        status_url = response.headers["Location"]
        resp = wait_for_async_resolution(client, status_url)
        data = resp.json()
    return data

# 0. to test outside of MS SQL just initialize the InputDataSet
# InputDataSet = pd.read_csv("../SampleData/LendingClub/10K_Lending_Club_LoansTrain.csv",nrows=10000)
# InputDataSet["API_KEY"] = "n-HQz5v5xAAi2N9AjwNkhBBxd97gvvKx"
# InputDataSet["USERNAME"]= "felix.huthmacher@datarobot.com"
# InputDataSet["WORKERCOUNT"] = "8"
# InputDataSet["TARGET"]= "is_bad"
# InputDataSet["DEPLOYMENT_ID"] = "None"
# InputDataSet["RETRAIN"] = "1"
# InputDataSet["PROJECTNAME"] = "DataRobot ML with MS SQL"
# InputDataSet["MODELMANAGEMENTENDPOINT"] =  "https://app.datarobot.com/api/v2"
# InputDataSet["PREDICTIONSENDPOINT"] =  "https://datarobot-support.orm.datarobot.com/predApi/v1.0"
# InputDataSet["DR_KEY"] = "e2225da5-1eaa-95de-7f48-36f4b2975bbb"
# InputDataSet["INSTANCE_ID"] =  "5b56f5308daae3002166d290"
# InputDataSet["predictionValue"] = ""
# InputDataSet["predictionLabel"] = ""
# InputDataSet["prediction"] = ""
# InputDataSet["predictionDate"] = ""

# 1. Get the DR client settings and authenticate
API_TOKEN = InputDataSet["API_KEY"].iloc[0]
USERNAME = InputDataSet["USERNAME"].iloc[0]
WORKERCOUNT = InputDataSet["WORKERCOUNT"].iloc[0]
PROJECTNAME = InputDataSet["PROJECTNAME"].iloc[0]
TARGET = InputDataSet["TARGET"].iloc[0]
DR_KEY = InputDataSet["DR_KEY"].iloc[0]
INSTANCE_ID = InputDataSet["INSTANCE_ID"].iloc[0]
MODELMANAGEMENTENDPOINT = InputDataSet["MODELMANAGEMENTENDPOINT"].iloc[0]
PREDICTIONSENDPOINT = InputDataSet["PREDICTIONSENDPOINT"].iloc[0]
DEPLOYMENT_ID = InputDataSet["DEPLOYMENT_ID"].iloc[0]
RETRAIN = InputDataSet["RETRAIN"].iloc[0]

MODELMANAGEMENTHEADERS = {"Content-Type": "application/json", "Authorization": "token %s" % API_TOKEN}
PREDICTIONSHEADERS = {"Content-Type": "application/json; charset=UTF-8","datarobot-key": "%s" % DR_KEY}

drclient = dr.Client(endpoint=MODELMANAGEMENTENDPOINT, token=API_TOKEN, ssl_verify=False)


# 2. get the data and do any data preparation / enrichment as required
#    in this tutorial no data preparation is required, we will just remove columns that are not required for the model
idata = InputDataSet
```

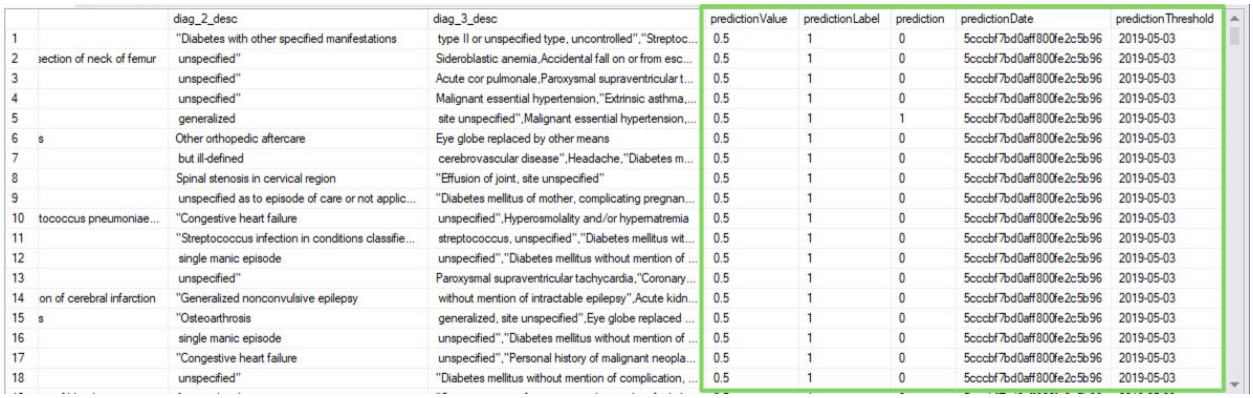
```
idata =
idata.drop(["API_KEY","USERNAME","DEPLOYMENT_ID","WORKERCOUNT","RETRAIN","TARGET","PROJECTNAME","DR_KEY","INST
ANCE_ID","MODELMANAGEMENTENDPOINT","PREDICTIONSENDPOINT","predictionValue","predictionLabel",
"prediction","predictionDate","predictionThreshold"], axis=1)

# 3. Check if deployment already exists, if not create one, else use the existing one
if str(DEPLOYMENT_ID) == "None" or str(DEPLOYMENT_ID) == "":
        # create new project & deployment
        print("creating new project with autopilot")
        newProject = dr.Project.start(sourcedata=idata, project_name =PROJECTNAME, target = TARGET, autopilot_on=True)
        newProject.set_worker_count(int(WORKERCOUNT))
        newProject.wait_for_autopilot()
        print("waiting for autopilot")
        recommendation_type = dr.enums.RECOMMENDED_MODEL_TYPE.RECOMMENDED_FOR_DEPLOYMENT
        recommendation = dr.models.ModelRecommendation.get(newProject.id, recommendation_type)
        bestModelId = recommendation.model_id
        crossvalidation = requests.post("%s/projects/%s/models/%s/crossValidation" % (MODELMANAGEMENTENDPOINT,
newProject.id, bestModelId), headers=MODELMANAGEMENTHEADERS )
        print(crossvalidation)
        payload = {
    "projectId": str(newProject.id),
    "modelId": str(bestModelId),
    "instanceId": str(INSTANCE_ID),
    "label": "DataRobot ML with MS SQL"+ str(date.today()),
    "description": "DataRobot ML with MS SQL"+ str(date.today()),
    "status": "active",
    "deploymentType": "dedicated",
    "trainingDataSubset": "eda"
  }
        deploymentresponse = drclient.post(
                        "/modelDeployments/asyncCreate",
                        data=payload,
                        headers={"Content-Type": "application/json"}
                )
        print(str(deploymentresponse))
        deployment_response = wait_for_result(drclient, deploymentresponse)
        DEPLOYMENT_ID = str(deployment_response["id"])
        print(str(DEPLOYMENT_ID))

else:
        print("use existing deployment: " + DEPLOYMENT_ID)

# 4. check if model retraining is required
if RETRAIN == "1":
  print("retrain model first before making predictions")
  # a. get current model of deployment
  deployment = requests.get("%s/modelDeployments/%s" % (MODELMANAGEMENTENDPOINT,DEPLOYMENT_ID),
headers=MODELMANAGEMENTHEADERS)
  model = dr.Model.get(deployment.json()["project"]["id"],deployment.json()["model"]["id"])
  # b. create new project with new training data
  retrainProject = dr.Project.start(sourcedata=idata, project_name=PROJECTNAME + str(date.today()) , target=TARGET,
autopilot_on=False)

  # c. train new model with training data
  modelJobId = retrainProject.train(model.blueprint_id)
  newModel = dr.models.modeljob.wait_for_async_model_creation(project_id=retrainProject.id, model_job_id=modelJobId)
  fi = newModel.get_or_request_feature_impact(600)

  # d. update deployment with new model
  model_Update = requests.patch("%s/modelDeployments/%s/model" % (MODELMANAGEMENTENDPOINT,DEPLOYMENT_ID),
headers=MODELMANAGEMENTHEADERS, data="{\"modelId\":\"%s\"}" % (newModel.id))

# 5. make predictions
print("make predictions with existing model")
predictions = pd.DataFrame([])
```

```
predictionResults = requests.post("%s/deployments/%s/predictions" % (PREDICTIONSENDPOINT,DEPLOYMENT_ID),
data=idata.to_json(orient="records"),auth=(USERNAME, API_TOKEN), headers=PREDICTIONSHEADERS)

#map results to a dataframe and fail gracefully if no predictions got returned
if "message" in predictionResults:
        print ("error during game result prediction: " + str(predictionResults["message"]))
if predictionResults.status_code == 200:
   items = json.loads(predictionResults.text)["data"]
   for item in items:
      rowId = item["rowId"]
      prediction = item["prediction"]
      predictionThreshold = item["predictionThreshold"]
      predictionLabel =  item["predictionValues"][0]["label"]
      predictionValue =  item["predictionValues"][0]["value"]

      predictions = predictions.append({"rowId": rowId, "prediction": prediction, "predictionThreshold": predictionThreshold,
"predictionLabel": predictionLabel, "predictionValue": predictionValue }, ignore_index=True)

#merge results with source
sourcedata = idata.reset_index()
sourcedata["rowId"] = sourcedata.index
output = pd.merge(sourcedata,predictions,left_on = ["rowId"], right_on = ["rowId"],how="left")
output["DEPLOYMENT_ID_NEW"] = str(DEPLOYMENT_ID)
output["predictionDate"] = str(date.today())
output = output.drop(["rowId","index"], axis=1)
print(list(output))
OutputDataSet = output';
DECLARE @SQL NVARCHAR(MAX)
IF object_id('dbo.#admissions') IS NOT NULL
BEGIN
        SET @SQL = N' SELECT *, ' + CONVERT(nvarchar(max), @RETRAIN) + ' as RETRAIN FROM dbo.#admissions JOIN
dbo.DRconfiguration ON 1 = 1;'
END
ELSE
BEGIN
        SET @SQL = N' SELECT *, ' + CONVERT(nvarchar(max), @RETRAIN) + ' as RETRAIN FROM dbo.admissions JOIN
dbo.DRconfiguration ON 1 = 1;'
END
DELETE FROM @PREDICTIONS
INSERT INTO @PREDICTIONS
EXEC sp_execute_external_script
  @language=N'Python'
, @script = @PSCRIPT
, @input_data_1 = @SQL

UPDATE a
SET prediction = p.prediction,
   predictionValue = p.predictionValue,
        predictionLabel = p.predictionLabel,
        predictionDate = p.predictionDate,
        predictionThreshold = p.predictionThreshold
FROM admissions a
JOIN @PREDICTIONS p
ON a.id = p.id

DECLARE @DEPLOYMENT_ID nvarchar(500)
SELECT TOP 1 @DEPLOYMENT_ID = DEPLOYMENT_ID FROM @PREDICTIONS
IF @DEPLOYMENT_ID IS NOT NULL
BEGIN
        UPDATE DRconfiguration
                SET DEPLOYMENT_ID =  CONVERT(nvarchar(500),@DEPLOYMENT_ID)
END

END
```

The procedure can actually be used for the initial model training and periodic retraining of the model, as well as the individual predictions, because based on the retrain parameter setting, it will either do a retraining of the model first, or generate the predictions directly.
You can test the procedure by simply running

```
EXEC sp_predictAdmissions
```

The result should look similar to the below.



		diag_2_desc	diag_3_desc	predictionValue	predictionLabel	prediction	predictionDate	predictionThreshold
1		"Diabetes with other specified manifestations	type II or unspecified type, uncontrolled",'Streptoc...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
2	ection of neck of femur	unspecified"	Sideroblastic anemia,Accidental fall on or from esc...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
3		unspecified"	Acute cor pulmonale,Paroxysmal supraventricular t...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
4		unspecified"	Malignant essential hypertension,"Extrinsic asthma,...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
5		generalized	site unspecified",Malignant essential hypertension,...	0.5	1	1	5cccbf7bd0aff800fe2c5b96	2019-05-03
6	s	Other orthopedic aftercare	Eye globe replaced by other means	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
7		but ill-defined	cerebrovascular disease",Headache,"Diabetes m...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
8		Spinal stenosis in cervical region	"Effusion of joint, site unspecified"	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
9		unspecified as to episode of care or not applic...	"Diabetes mellitus of mother, complicating pregnan...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
10	tococcus pneumoniae...	"Congestive heart failure	unspecified",Hyperosmolality and/or hypernatremia	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
11		"Streptococcus infection in conditions classifie...	streptococcus, unspecified","Diabetes mellitus wit...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
12		single manic episode	unspecified","Diabetes mellitus without mention of ...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
13		unspecified"	Paroxysmal supraventricular tachycardia,"Coronary...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
14	on of cerebral infarction	"Generalized nonconvulsive epilepsy	without mention of intractable epilepsy",Acute kidn...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
15	s	"Osteoarthrosis	generalized, site unspecified",Eye globe replaced ...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
16		single manic episode	unspecified","Diabetes mellitus without mention of ...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
17		"Congestive heart failure	unspecified","Personal history of malignant neopla...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03
18		unspecified"	"Diabetes mellitus without mention of complication, ...	0.5	1	0	5cccbf7bd0aff800fe2c5b96	2019-05-03

Create SQL trigger

The last step is to create a SQL trigger, which invokes the stored procedure every time a new admission is added to the admissions table.

```
CREATE TRIGGER admissions_after_insert
ON admissions
AFTER INSERT
AS
BEGIN
        SET NOCOUNT ON
        -- pass new values to stored procedure via temp table
        SELECT *
        INTO #admissions
  FROM INSERTED
        EXEC [dbo].[sp_predictAdmissions] 0
END;
```

Conclusion

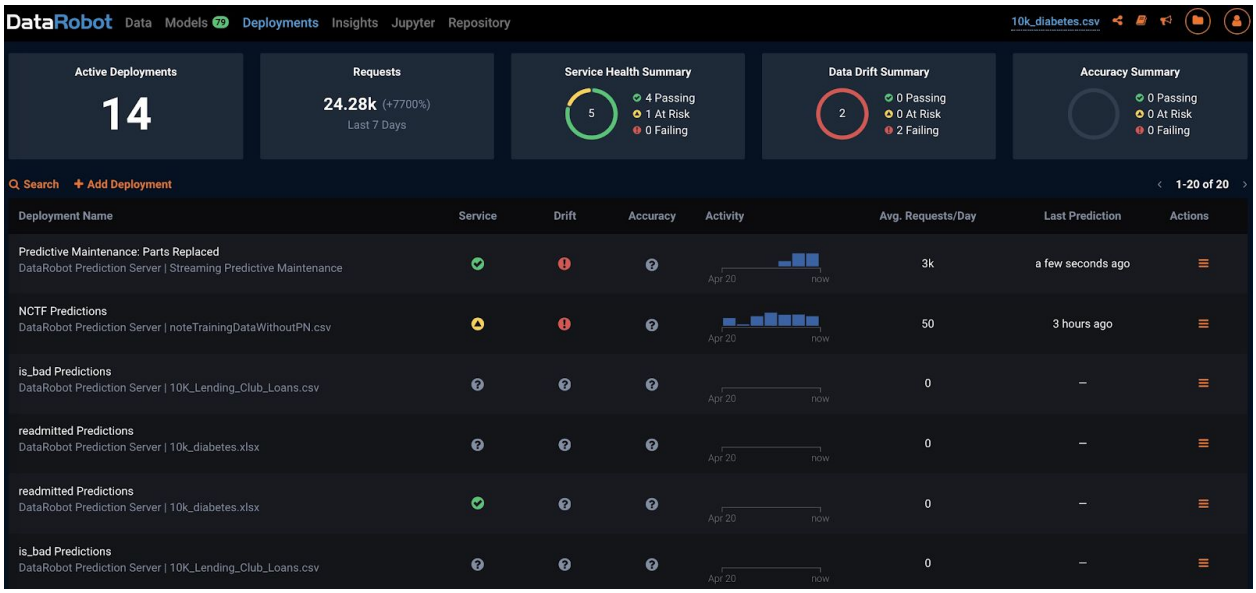
Let's recap what we have done so far:

1) We created a SQL stored procedure which functions as a wrapper for our python script, so that we can train a model and make predictions from within SQL server
2) We created a sql trigger, which generates a new prediction every time a new record is added to the table.

Based on the above work, the prediction results can be consumed in a production application, such as a Tableau dashboard or an EMR application, the same way as any of the other data points.
Also, since everything is instigated from within MS SQL, you can easily schedule a job to retrain the model periodically and thereby ensure a high model performance over time.

As a result, what used to take months and required Data Science knowledge and programming skills, as well as dedicated hardware and software, can now be done in an automated process with standard software and hardware within just one hour by one developer alone!

Another benefit of the DataRobot API is that it comes with an extensive UI and dedicated deployment dashboards, that allow you to track the not only basic usage metrics of your model such as the number of predictions or average response time, but also allows you to monitor your model health. This is useful to ensure high model performance and also to automate model retraining.



Next steps

While the above sample code is a very good starting point, you might want to enhance the functionality. Below I listed some pointers on what you might want to explore next:

1) Explore the API documentation and add error handling. For example, you can easily add email notification, if any error during the automated process occurs.
2) Create a job to utilize the Model Management Health and rather than retraining a model based on a fixed scheduled (e.g. monthly), trigger a model retraining if the model health deteriorates.
3) You might have noticed that we currently only predict the likelihood of a patient being readmitted. But there might be other features that you might want to predict (e.g. customer satisfaction, HepC vaccinations etc) as well. The DataRobot API makes it easy to train and predict multiple targets at the same time. This would only require minor modifications within the Python script above.
4) Explore model features and save specific model details within MS SQL Server table.

About the author

Felix Huthmacher is a Field Engineer at DataRobot

ML enthusiast, and hobby app developer, who has worked well over a decade in IT consulting in various industries.