Bringing AI into the Database with Db2's In-database Machine Learning

Data Server Day – Stockholm

September 27, 2023

Kelly Schlamb, Data & AI, IBM

kschlamb@ca.ibm.com

Agenda

- A brief look at machine learning concepts and terminology
- Db2’s in-database machine learning stored procedures
- Creating your own Python UDFs
- Troubleshooting Python UDFs
- Bringing external machine learning models into Db2 Python UDFs

What is AI & ML?

Data Science

The practice of applying various scientific and statistical methods, algorithms, approaches and processes…

using programming languages and software frameworks…

to extract knowledge, insights and recommendations from data…

and deliver them to business users and consumers in consumable applications

Artificial Intelligence – System that mimics human intelligence (very broad ranging applications, methods and supporting technologies)

Machine Learning – Enable machines to learn from data and make accurate predictions, without being explicitly programmed to do so

Deep Learning – Perform complex tasks (like speech and image recognition) by exposing multilayered neural networks to vast amounts of data

What is a Model?

- A mathematical representation of a real-world process
- Based on finding patterns in historic data
- The model can then make predictions against new data presented to it (this process is referred to as inferencing or scoring)
- It is built from a mathematical formula/algorithm (or combination of multiple ones), with parameters whose values need to be learned from the data
    - For example, given a data pattern that generally follows a straight line, what are the **M** and **B** parameter values in **Y = MX + B** that best describe the data?
- The process of learning and fitting a model to the data is referred to as training

Sample Scenario

Name	Age	Married	Income	Last Month Paid	Default
J. Smith	49	Yes	$208,267	Yes	False
A. Francis	21	No	$ 49,124	No	True
R. Black	64	Yes	$ 27,348	Yes	True
S. Tanner	56	Yes	$ 87,352	Yes	False

Is there a pattern to who defaults and who doesn't?

Can you build a model to predict this for new data?

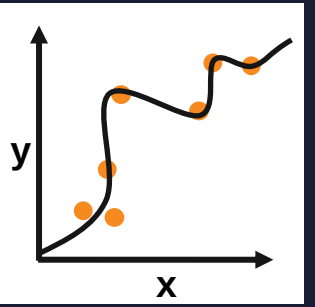
Name	Age	Married	Income	Last Month Paid	Default
J. Austin	27	Yes	$178, 652	Yes	?
P. Turner	48	No	$ 24,391	No	?
K. Wayne	64	Yes	$ 77,736	Yes	?

Common Machine Learning Techniques

Regression (supervised learning)

- Predict a quantity/continuous value
- Linear regression, polynomial regression, …

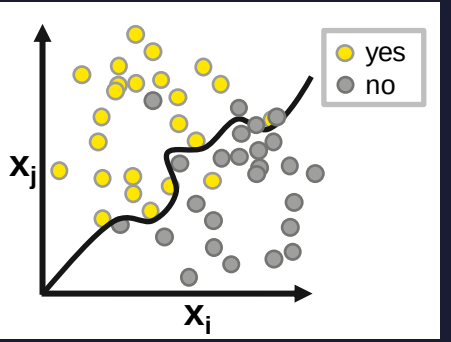
What will my future sales look like?



Classification (supervised learning)

- Predict a label/category
- Binary or multiclass

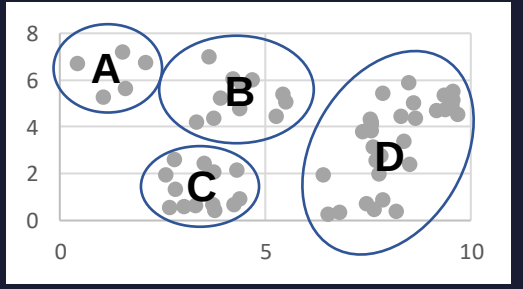
Will this client default on their loan?



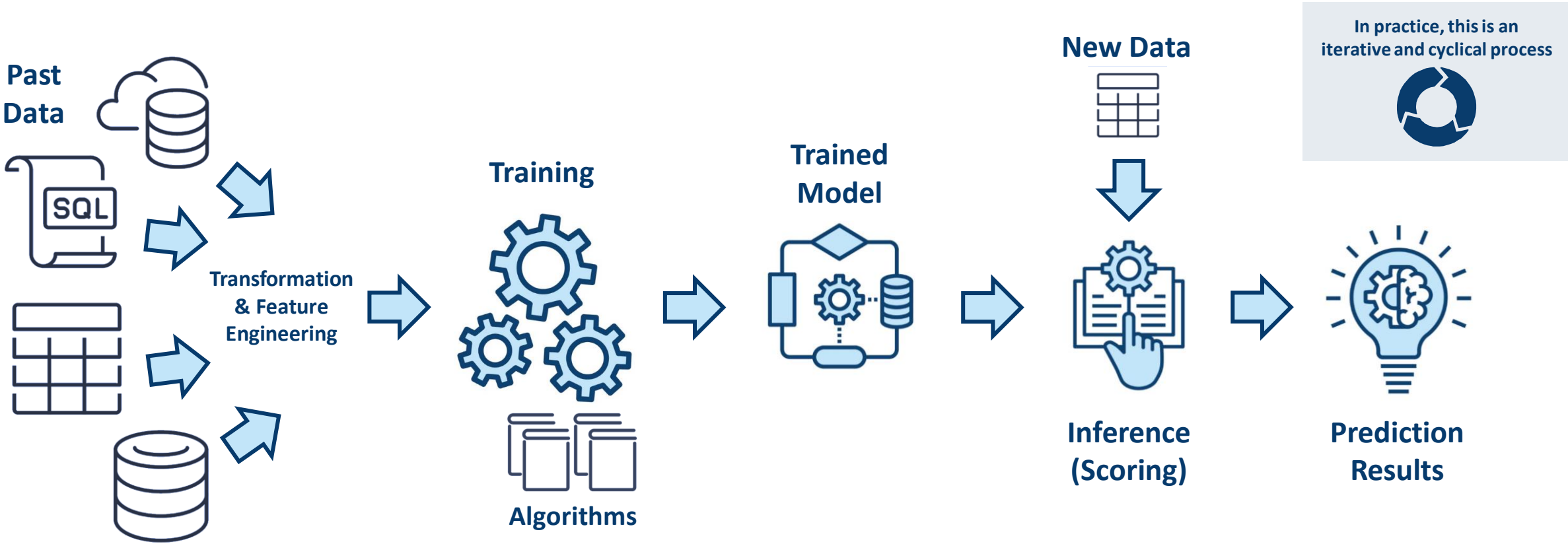
Clustering (unsupervised learning)

- Group similar objects together

Are there similarities between visitors to our website?



Machine Learning Workflow



Training Phase

Inference Phase

Db2 In-Database Machine Learning Stored Procedures

Db2 Python UDFs

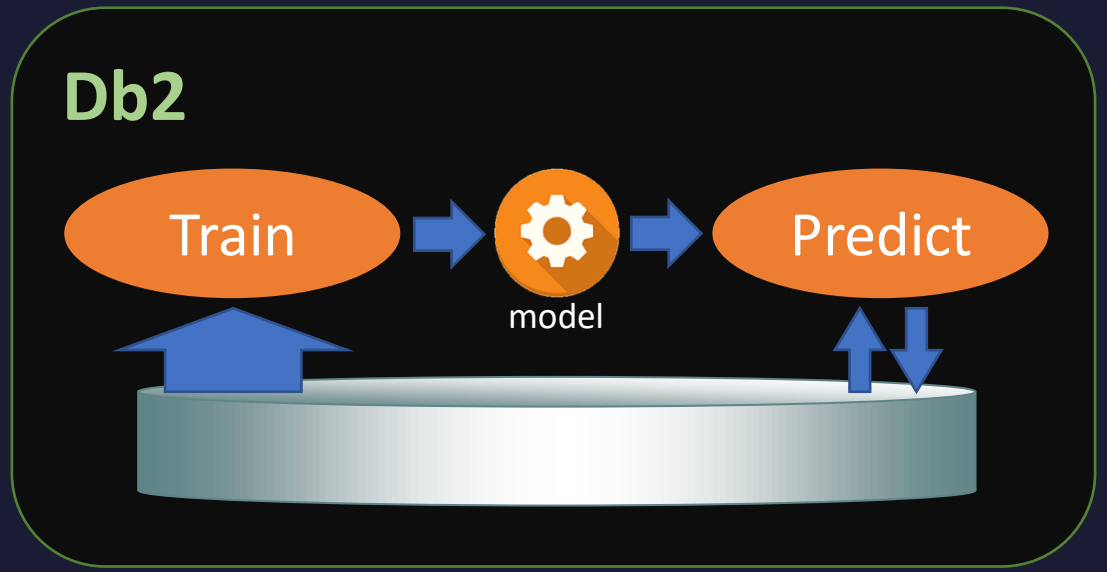
IBM Db2

Why In-Database Machine Learning?

- Avoids data movement (no governance, bandwidth, latency issues)
- Real-time inferencing
- Simple (less complex, less environments => cost benefits)
- Data quality inherently high due to your efforts
- Database capabilities can be exploited
  - Built-in access control and security
  - Typically running on a high-performing server
  - Distributed processing
  - SQL access for building models and/or inferencing

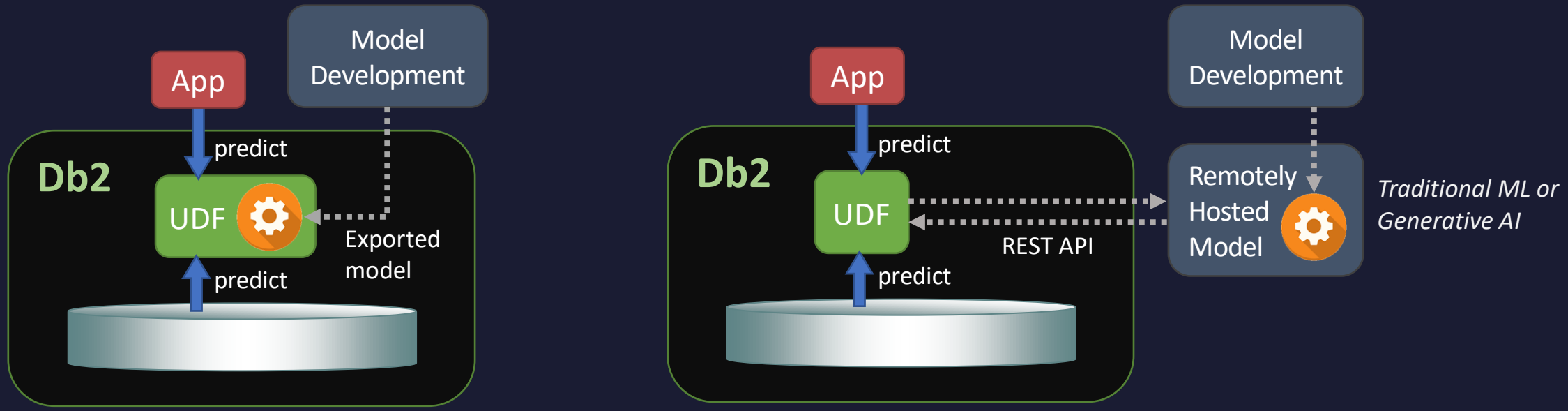
Benefits of In-database Machine Learning Stored Procedures

- Requires no model development experience or external machine learning development environment – the algorithms are built into Db2
- Training where the data lives – no need to transfer large volumes of data elsewhere
- Scalability – exploit multi-node training via Db2 DPF
- Low-latency predictions, inferencing where the data lives



Benefits of Machine Learning with Python UDFs

- Train in your model development environment of choice
  - Training can be accelerated via GPUs
- Allows for transformations and algorithms not found natively in Db2
- Low-latency predictions, inferencing where the data lives
- Predict (in-DB, call from app) via SQL
- Incorporate into data entry workflow through triggers (insert and score)



Built-in Machine Learning Stored Procedures in Db2

Algorithms

- Decision Tree (classification)
- Naïve Bayes (classification)
- Linear Regression (regression)
- K-Means (clustering)

Tasks

- Data Exploration
- Data Transformation
- Model Building
- Model Tuning
- Model Evaluation
- Model Inferencing
- Model Management
- Troubleshooting

https://www.ibm.com/docs/en/db2/11.5?topic=content-in-database-machine-learning

Let’s explore the stored procedures through a simple, common ML sample use case…

Photo by Frank Mayfield,
CC BY-SA 2.0 <https://creativecommons.org/licenses/by-sa/2.0>, via Wikimedia Commons

Iris Species

Sepal Length	Petal Length
Sepal Width	Petal Width

Setosa?
Versicolor?
Virginica?

Training Data

iris_species.csv

```
sepal_length,sepal_width,petal_length,petal_width,species
5.1,3.5,1.4,0.2,setosa
4.9,3.0,1.4,0.2,setosa
4.7,3.2,1.3,0.2,setosa
…
7.0,3.2,4.7,1.4,versicolor
6.4,3.2,4.5,1.5,versicolor
6.9,3.1,4.9,1.5,versicolor
…
6.3,3.3,6.0,2.5,virginica
5.8,2.7,5.1,1.9,virginica
7.1,3.0,5.9,2.1,virginica
…
```

https://ibm.box.com/v/iris-species

Database Setup for ML Stored Procedures

- *Linux on x86 or Power*
- *Limited support for zLinux*

Enables creation of in-database ML stored procedures

16K page size user and system temporary table spaces required

```
db2set DB2_ENABLE_ML_PROCEDURES=YES
db2 create database ml_db using codeset utf-8 territory US pagesize 16384
db2 connect to ml_db
db2 create regular tablespace ml_tbspc
db2 "call sysinstallobjects('IDAX', 'C', 'ML_TBSPC', null)"
db2 grant use of tablespace ml_tbspc to public
db2 connect reset
```

Must be a REGULAR table space

Procedure caller must have USE privilege on the table space

Creates analytics procedures

Importing Training Data

A unique row identifier is required

```
create table iris (id int not null generated always as identity,
                   sepal_length  dec(2,1),
                   sepal_width   dec(2,1),
                   petal_length  dec(2,1),
                   petal_width   dec(2,1),
                   species_name  char(10))

import from iris_species.csv of del skipcount 1 insert into iris
  (sepal_length, sepal_width, petal_length, petal_width, species_name)
```

Skip the first row if there's a header in the input data set

Exploring the Data Set

```
call idax.summary1000('intable=iris, outtable=iris_summary')
select * from iris_summary
select * from iris_summary_num
select * from iris_summary_char
```

Generates multiple tables

<OUTTABLE>

For each of first 1000 columns (where applicable):

- Column name
- Distinct values
- Most frequent value
- Most frequent cases
- Average
- Standard deviation
- Minimum
- Maximum
- …

<OUTTABLE>_NUM

Numeric column information
(table created only if numeric columns)

<OUTTABLE>_TIME

Time column information
(table created only if time columns)

<OUTTABLE>_CHAR

Character column information
(table created only if character columns)

<OUTTABLE>_TIMESTAMP

Timestamp column information
(table created only if timestamp columns)

<OUTTABLE>_DATE

Date column information
(table created only if date columns)

Handling Missing Values (Data Imputation)

How is a model's accuracy impacted by data with missing values?

Should you just delete the rows (or maybe the columns) altogether?

A common approach is to substitute missing values with intelligent replacement values **(imputation)**

sepal_length	sepal_width	petal_length	petal_width	class_name
5.1	3.5	1.4	0.2	setosa
7.0	*NULL*	3.7	1.4	versicolor
6.3	3.3	6.0	*NULL*	virginica
6.4	3.2	4.5	1.5	versicolor
4.9	3.0	1.4	0.2	setosa
6.9	3.1	*NULL*	1.5	versicolor
4.7	3.2	1.3	0.2	setosa

Options for replacement:

- Mean of column values
- Median of column values
- Most frequently seen value
- Default of your choosing

Handling Missing Values (Data Imputation) (cont.)

```
call idax.impute_data('intable=iris,
                       method=mean,
                       outtable=iris_mean');

call idax.impute_data('intable=iris,
                       method=freq,
                       outtable=iris_freq');

call idax.impute_data('intable=setosa_view,
                       method=median,
                       outtable=setosa_median');

call idax.impute_data('intable=iris,
                       method=replace,
                       incolumn=sepal_length,
                       numericValue=2,
                       outtable=iris_replace');
```

Input table

Type of substitution method

Output table

Input can be a view as well!

Specify a list of columns to perform substitution on (all columns by default)

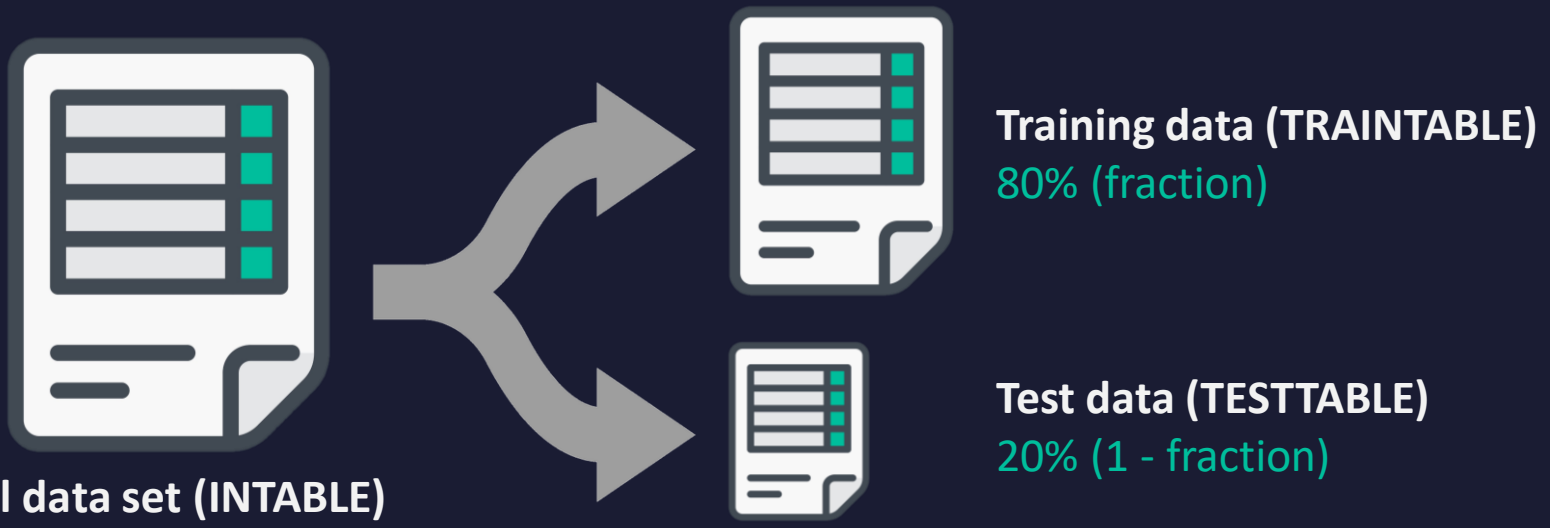
You can specify the specific value to substitute in

Splitting Data Set into Training and Test

```
call idax.split_data('intable=IRIS, traintable=IRIS_TRAIN,
                      testtable=IRIS_TEST, id=ID,
                      fraction=0.80, seed=1')
```

Percentage of input data that will be used in training data

Specify a seed if you want repeatable results

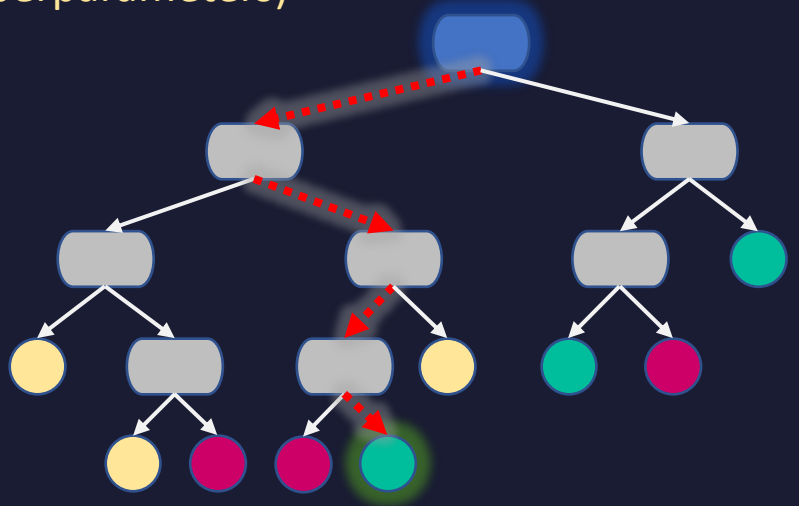


Training a Decision Tree Model

```
call idax.grow_dectree('model=IRIS_TREE_MODEL, intable=IRIS_TRAIN,
                        id=ID, target=SPECIES_NAME,
                        minimprove=0.02, minsplit=3, maxdepth=10')
```

```
call idax.print_model('model=IRIS_TREE_MODEL')

------------------------------------------------------------
 -- decision tree model: "DB2INST1"."IRIS_TREE_MODEL" --
  PETAL_LENGTH <= 1.9E0
   | if true then class -> setosa
   | PETAL_WIDTH <= 1.7E0
   | | PETAL_LENGTH <= 4.9E0
   | | | PETAL_WIDTH <= 1.6E0
   | | | | if true then class -> versicolor
   | | | | if false then class -> virginica
   | | | PETAL_WIDTH <= 1.5E0
   | | | | if true then class -> virginica
   | | | | if false then class -> versicolor
   | | PETAL_LENGTH <= 4.8E0
   | | | SEPAL_LENGTH <= 5.9E0
   | | | | if true then class -> versicolor
   | | | | if false then class -> virginica
   | | | if false then class -> virginica
```

Adjustable
training parameters
(hyperparameters)



Evaluating the Model (Confusion Matrix Example)

```
call idax.confusion_matrix('intable=IRIS_TEST, id=ID, target=SPECIES_NAME,
                            resulttable=IRIS_RESULT, resultid=ID,
                            resulttarget=CLASS, matrixtable=IRIS_CMATRIX')
```

```
select * from iris_cmatrix

REAL       PREDICTION CNT
---------- ---------- -----------
setosa     setosa              11
setosa     versicolor           1
versicolor versicolor           7
versicolor virginica            1
virginica  setosa               1
virginica  versicolor           1
virginica  virginica            8
```

Use this information to calculate model accuracy, precision, recall, F1, etc.

Real/Actual Class

Predicted Class	setosa	versicolor	virginica
setosa	11	0	1
versicolor	1	7	1
virginica	0	1	8

Confusion Matrix

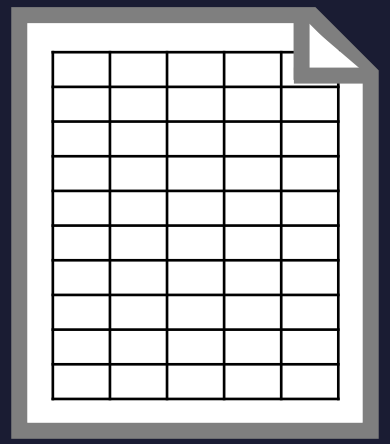
Making Predictions (a.k.a. Inferencing or Scoring)

```
call idax.predict_dectree('model=IRIS_TREE_MODEL,
                           intable=IRIS_TEST, outtable=IRIS_RESULT, id=ID
                           prob=true, outtableprob=IRIS_PROB')
```

```
select * from iris_result order by id

ID  CLASS          PROB
--- -------------- ----------------------
  4 setosa         +1.00000000000000E+000
 11 setosa         +9.58000000000000E+001
 12 setosa         +1.00000000000000E+000
 …
 55 versicolor     +9.75000000000000E-001
 61 versicolor     +9.64000000000000E-001
 64 versicolor     +9.75000000000000E-001
 …
108 virginica      +9.72972972972973E-001
130 virginica      +6.00000000000000E-001
141 virginica      +9.72972972972973E-001
```

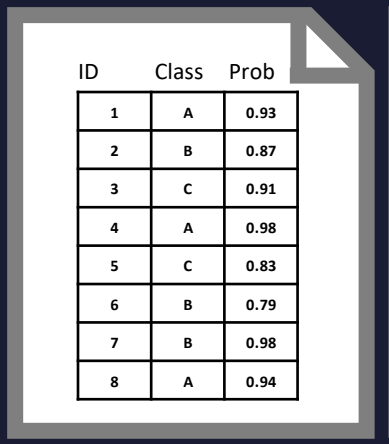
Include probability along with prediction in results table

Create a separate table for probabilities (for all classes, for each input row)

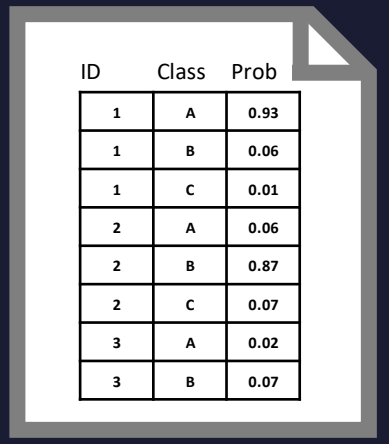


INTABLE

Predict



OUTTABLE



OUTTABLEPROB

Troubleshooting & Cleanup

The CALL statement for ML procedures can return a generic SQL error message; this variable contains the specific CDFAA##### error code

Troubleshooting & Error Handling

```
values idax.last_message_code
values idax.last_message
```

Text for error message associated with that CDFA error message code

Cleanup

```
call idax.drop_model('model=IRIS_TREE_MODEL')
call idax.drop_summary1000('intable=iris_summary');
drop table <otherGeneratedTables>
```

Python UDFs – The Basics

- Linux-only
- No documentation of Python versions supported (that I could find), but Python 3.6+ has worked in all of my testing
- Function types: Scalar (UDSF), Table (UDTF), Aggregate (UDAF)
- UDF must be defined as `fenced`
- DPF/pureScale: Copy Python source file (.py) to same location on each node
- Set `PYTHON_PATH` to your Python runtime executable
  - e.g. `db2 update dbm cfg using python_path /usr/bin/python3`
  - Python executable must be accessible by the fenced user ID

Simple Scalar UDF

Create a function to add 2 integers

/home/db2inst1/PythonUDFs/add.py

```
import nzae

class add(nzae.Ae):
    def _getFunctionResult(self, row):
        var1, var2 = row
        return var1 + var2

add.run()
```

Language is Python

Parameter passing style for Python UDFs

Must be fenced

Fully qualified location of .py source file

```
create function add_udf(integer, integer)
  returns integer
  language python
  parameter style npsgeneric
  returns null on null input
  fenced
  no sql
  external name '/home/db2inst1/PythonUDFs/add.py'

values add_udf(14, 23)    -- Sample UDF call
```

A Few Things That Tripped Me Up…

These are all function execution errors (returned from CALL)

```
SQL1646N  A routine failed because the fenced user ID cannot access required
files in the sqllib directory or other instance or database directories.
```

PYTHON_PATH incorrectly set

```
SQL0443N  Routine "<UDF>" (specific name "") has returned an error SQLSTATE
with diagnostic text "<class 'FileNotFoundError: [Errno 2] No such file or
director".  SQLSTATE=38N93
```

Invalid path specified in CREATE FUNCTION

```
SQL0443N  Routine "<UDF>" (specific name "") has returned an error SQLSTATE
with diagnostic text "<class 'PermissionError: [Errno 13] Permission denied:
'/home".  SQLSTATE=38N93
```

Fenced user doesn't have access to Python source file

```
SQL0443N  Routine "<UDF>" (specific name "") has returned an error SQLSTATE
with diagnostic text "<MESSAGE>".  SQLSTATE=38N93
```

Various coding errors. See routine.#.log and db2diag.log for more information, stacks, …

Db2's Routine Log

- `<DIAGPATH>/routinelog/routine.#.log`
- Exceptions/stack tracebacks recorded in this log file
- Use `self.log(<string>)` to log your own messages to the log file

```
self.log("My debug message: var1=" + str(var1)
          + " (line " + str(sys._getframe().f_lineno) + ")")
```

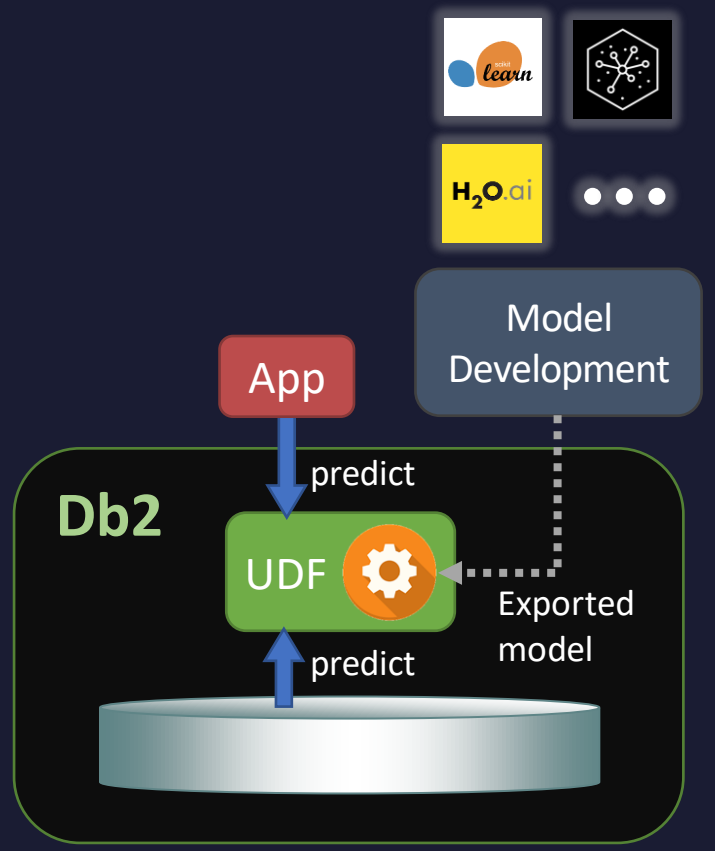
```
2023-02-22-21.24.25.574870-300 I6859E502              LEVEL: Info
PID     : 80013                TID : 140344167520064 PROC : python3
INSTANCE: db2inst1             NODE : 000            DB   : ML_DB
APPID   : *LOCAL.db2inst1.210223015858
HOSTNAME: ubuntu
FUNCTION: DB2 UDB, routine_infrastructure, sqlerRoutineLogMessage, probe:194
DATA #1 : String, 76 bytes
{USERNAME: DB2INST1 ROUTINEID: 67483 ROUTINENAME: DB2INST1.IRIS_PREDICT}
DATA #2 : String, 35 bytes
My debug message: var1=43 (line 24)
```

Troubleshooting Hints & Tips

- Ensure DBM CFG parameter `PYTHON_PATH` set correctly
- Python and packages must be installed on all nodes (DPF, pureScale)
- Verify Python script is accessible (by instance owner or fenced user if different)
- Log messages to Db2's routine log (`self.log("<string>")`) or use other Python methods for writing messages and data to your own log files (use absolute names)
- Consider writing a simple standalone Python application with your inferencing logic to ensure it works correctly prior to creating a UDF

Embedding Machine Learning Models in a Python UDF

- Generally speaking, if you can load a model's runtime code into a Python application then you can use it within a Db2 Python UDF
- May require installing package/module dependencies (per the vendor/framework's requirements)
- I've successfully tested models built using:
  - scikit-learn within a Jupyter Notebook
  - Watson Studio's AutoAI in IBM Cloud Pak for Data
  - H2O Driverless AI (via their MOJO model runtime format)
- I've developed models on Linux x86 and used with Db2 running on both Linux x86 and Linux ppc64le (Power)



Training a Model in Scikit-learn

Decision Tree for Db2 Python UDF - Iris Flower Species

Notebook Summary

The Iris dataset is a classic and very easy to understand multiclass classification dataset. Based on four measurements, one can predict the species of Iris.

This notebook will create a decision tree model to perform that prediction. At the end of the notebook the model will be exported/dumped to disk such that it can be used within a Db2 Python UDF for in-database scoring purposes.

Scikit-learn includes this Iris dataset, along with a function for loading the data:

Documentation: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_iris.html

- Classes: 3 (setosa, versicolor,virginica)
- Features: 4 (sepal_length, sepal_width, petal_length, petal_width)
- Samples per class: 50
- Samples total: 150

Step 1: Setup Environment

In [ ]:

```
# Required imports
import numpy as np
import os
import pandas as pd
from sklearn import tree
from sklearn.datasets import load_iris
from matplotlib import pyplot
from joblib import dump

# Ensure consistency across multiple runs
np.random.seed(1)
```

Step 2: Load and Display the Data

In [ ]:

```
# Load the iris dataset. By default it's being loaded as a Bunch object.
iris = load_iris()
```

- Scikit-learn is a free, open-source ML library for Python
- Includes several algorithms and model types
- Iris data set is built-in
- Download notebook: https://github.com/kschlamb/db2_machine-learning/tree/main/Python UDF-PreBuiltModel-IRIS

Loading the Data Set

Step 2: Load and Display the Data

In [2]:
```
# Load the iris dataset. By default it's being loaded as a Bunch object.
iris = load_iris()
```

In [3]:
```
# Show the feature names (input columns)
list(iris.feature_names)
```

Out[3]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

In [4]:
```
# Show the target names (the values we're trying to predict... i.e. the class names)
list(iris.target_names)
```

Out[4]:
```
['setosa', 'versicolor', 'virginica']
```

In [5]:
```
# Display the input data.
pd.DataFrame(iris.data, columns=iris.feature_names).head()
```

Out[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Training and Printing the Model

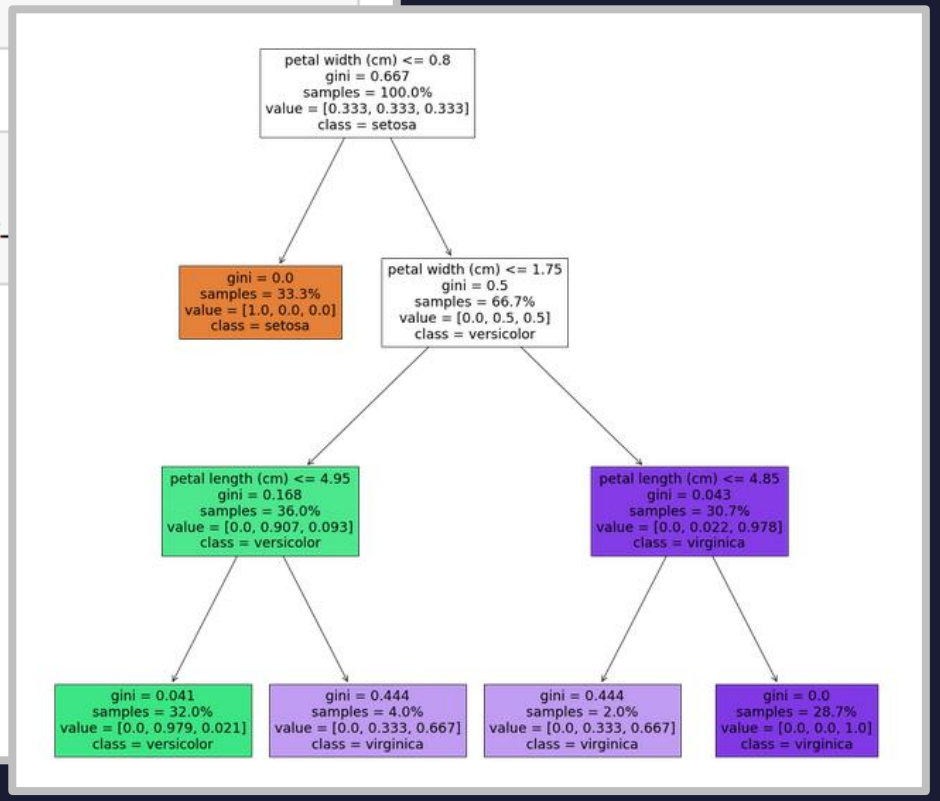
Step 3: Train Model and Display It

In [6]:
```
# Create and train the decision tree classifier model.

X = iris.data
y = iris.target

tree_model = tree.DecisionTreeClassifier(max_depth=3, random_state=1)
tree_model.fit(X, y)
```

Out[6]:
```
DecisionTreeClassifier(max_depth=3, random_state=1)
```

In [7]:
```
# Print textual representation of decision tree

tree_text = tree.export_text(tree_model, feature_names=iris.feature_
print(tree_text)
```

```
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.95
|   |   |   |--- class: 2
|   |--- petal width (cm) >  1.75
|   |   |--- petal length (cm) <= 4.85
|   |   |   |--- class: 2
|   |   |--- petal length (cm) >  4.85
|   |   |   |--- class: 2
```



Exporting the Model to Disk

Step 5: Export Model to Disk

In [11]:
```
# Export the model to disk locally (which can be re-loaded via the load() method).
dump(tree_model, 'iris_decision_tree_model.bin')
```

Out[11]:
```
['iris_decision_tree_model.bin']
```

In [12]:
```
# Export the target names (classes) so they can be used in the Db2 Python UDF.
dump(iris.target_names, 'iris_target_names.bin')
```

Out[12]:
```
['iris_target_names.bin']
```

- joblib.dump( ): Persists a Python object into a file (using Pickle object serialization)
- joblib.load( ): Reconstructs the Python object from the file (called from the Db2 UDF)

Db2 UDF Source Code

iris_udf.py

Import required modules

Derived from the nzae.Ae class

Load model

Main code body for the UDF

Make prediction

Return predicted class to the caller

```
import nzae
import numpy as np
from joblib import load
from sklearn.tree import DecisionTreeClassifier

class predict_iris_species(nzae.Ae):

    # As part of initialization, load the saved model
    # and the target class (species) names.
    def _setup(self):
        self.model = load('/home/db2inst1/Db2PythonUDF-IRIS/iris_decision_tree_model.bin')
        self.targets = load('/home/db2inst1/Db2PythonUDF-IRIS/iris_target_names.bin')

    # Main method for the Db2 UDF.
    def _getFunctionResult(self, row):

        # The input row is provided as a list (e.g. [1.0, 2.0, 3.0, 4.0],
        # which needs to be converted into a 2D array before calling the
        # model's predict() method. The result is a 1D array containing the
        # class ID.

        row_array = np.array(row).reshape(1, -1)
        predict_class_id = self.model.predict(row_array)[0]

        # Return the class name.

        predict_class_name = self.targets[predict_class_id]
        return(predict_class_name)

predict_iris_species.run()
```

Standardize where you place your exported models and how you manage them

Creating and Calling the UDF

```
create or replace function iris_udf(float, float, float, float)
  returns char(10)
  returns null on null input
  parameter style npsgeneric
  language python
  no sql
  external name '/home/db2inst1/Db2PythonUDF-IRIS/iris_udf.py';
```

```
values iris_udf(6.8, 3.1, 5.0, 2.3);

1
----------
virginica

  1 record(s) selected.
```

User-Defined Table Functions (multi-column results)

```
create or replace function iris_table_udf(float, float, float, float)
  returns table (species char(10), probability float)
  returns null on null input
  …
```

```
select * from table(iris_table_udf(5.6, 2.9, 3.6, 1.2))

SPECIES    PROBABILITY
---------- ------------------------
versicolor   +9.791666666666667E-001

  1 record(s) selected.
```

iris_table_udf.py

```
# Main method for the Db2 UDF.
def _getFunctionResult(self, row):
  …
  return(predict_class_name, predict_probability)
```

Automatic Inferencing Through Triggers

```
create table iris_new (sepal_length dec(2,1), sepal_width dec(2,1),
                       petal_length dec(2,1), petal_width dec(2,1),
                       predicted_species_name char(10))

create or replace trigger predict_iris_trigger
    before insert on iris_new
    referencing new as n
    for each row
        set n.predicted_species_name = (values (iris_udf(sepal_length, sepal_width,
                                                         petal_length, petal_width)))
```

Table with input columns and prediction column

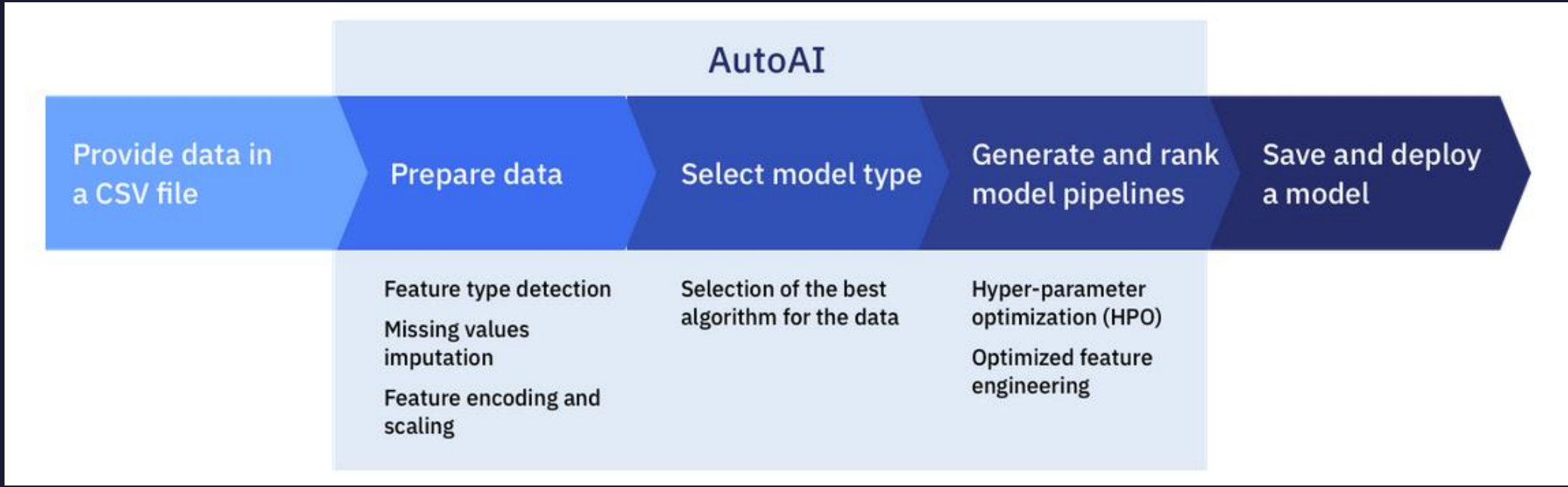
Trigger called when new row(s) added; fills in the prediction column by calling the UDF

```
insert into iris_new (sepal_length, sepal_width, petal_length, petal_width) values (4.8, 3.0, 1.4, 0.1)
insert into iris_new (sepal_length, sepal_width, petal_length, petal_width) values (5.2, 2.7, 1.7, 1.0)
insert into iris_new (sepal_length, sepal_width, petal_length, petal_width) values (6.4, 3.0, 5.8, 2.3)

select * from iris_new;

SEPAL_LENGTH SEPAL_WIDTH PETAL_LENGTH PETAL_WIDTH PREDICTED_SPECIES_NAME
------------ ----------- ------------ ----------- ----------------------
         4.8         3.0          1.4         0.1 setosa
         5.2         2.7          1.7         1.0 versicolor
         6.4         3.0          5.8         2.3 virginica

  3 record(s) selected.
```

Only input data provided

Results of this automatic inferencing/scoring

Watson Studio AutoAI



Build models faster
Automate data preparation and model development

Discover more use cases
Supercharge collaboration with AI everywhere to disrupt and transform

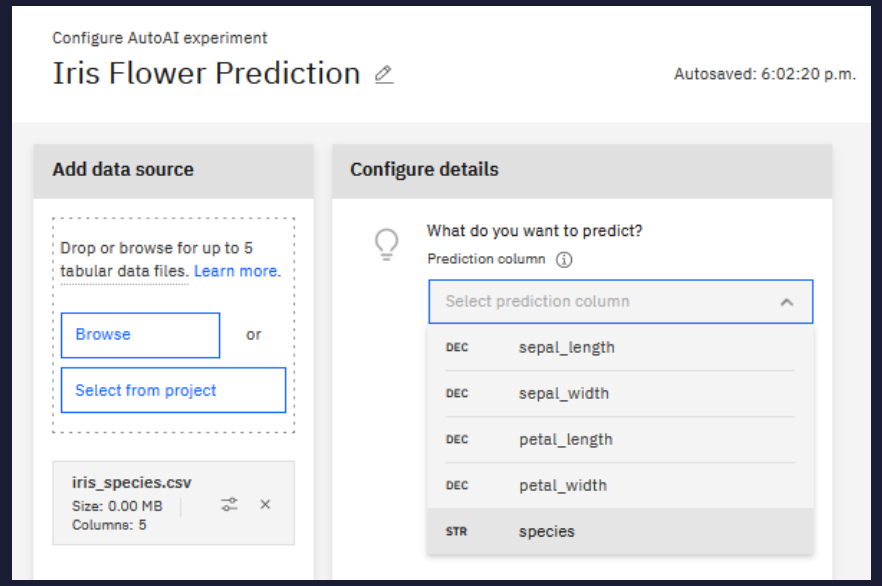
Rank and explore models
Quickly compare candidate pipelines to find the best model for the job

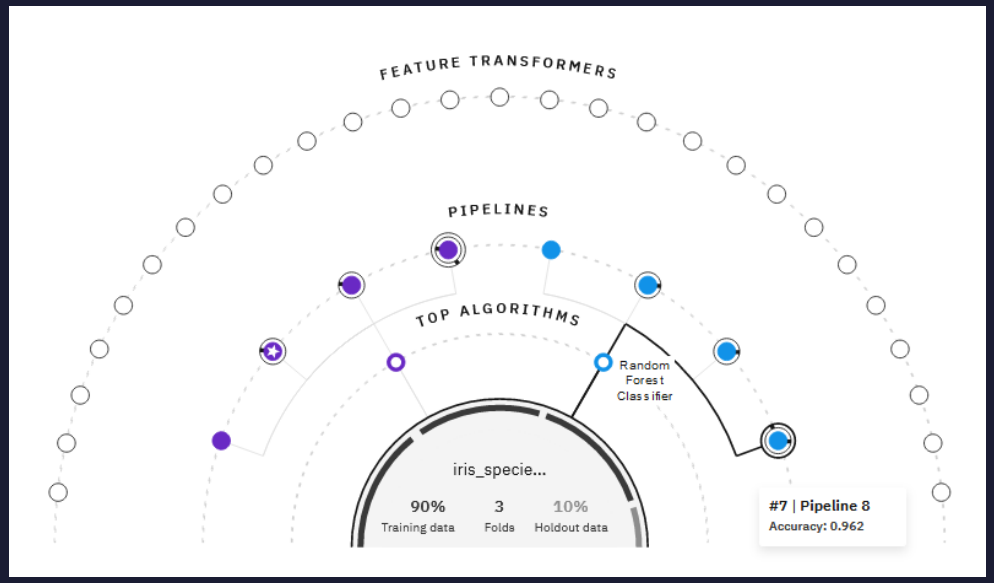
Jump the skills gap
No coding? No problem – get started with a couple clicks

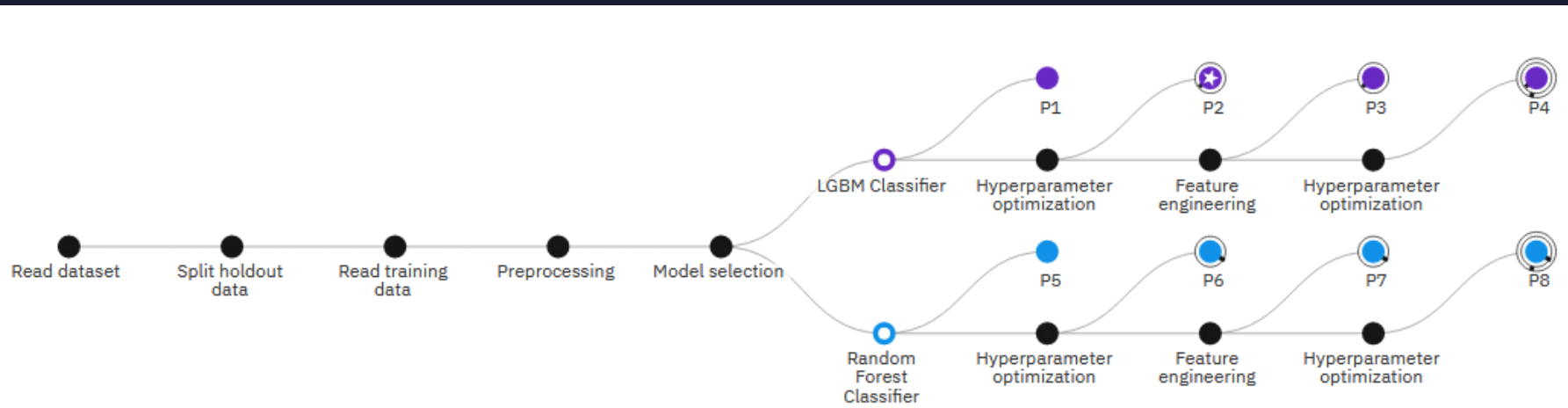
Find signal from noise
Auto-feature engineering makes it easy to extract more predictive power from your data

Ready, set, deploy
Pipelines generated with AutoAI can be deployed to REST APIs with one click

Using Watson Studio “AutoAI” Models in Db2







Run AutoAI experiment

Create notebook from experiment

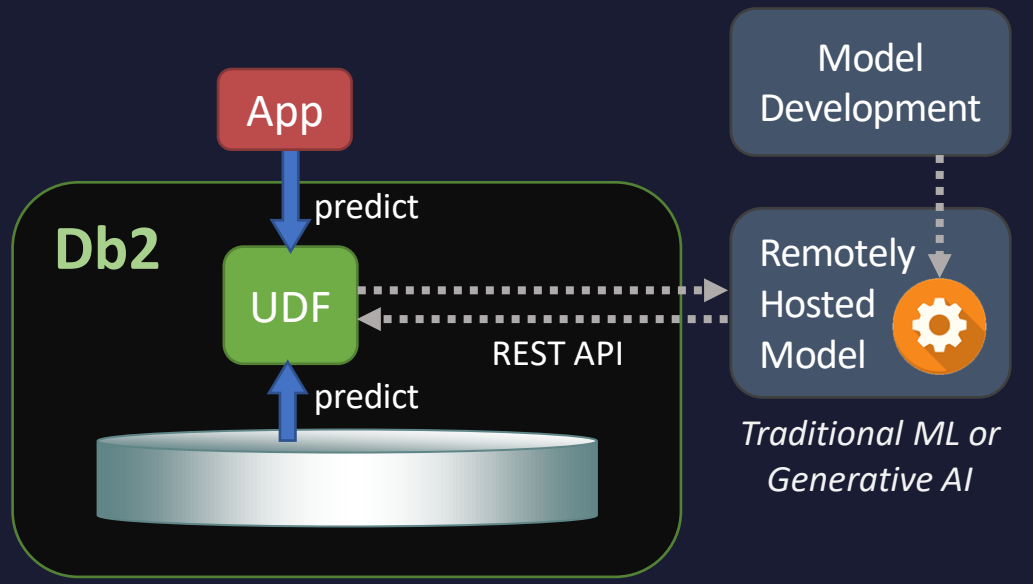
Modify & run notebook: Export model to obj storage

Download model from object storage

Load model into Db2 UDF

Remotely Hosted Models

- Models accessible via REST APIs can be called via a Db2 UDF
- Example call to Driverless AI model:




```
...
payload = { "id": 1,                  # Arbitrary identifier
            "method": "score",        # Single row scoring function
            "params": { "row": { "sepal_length": inSepalLength,
                                 "sepal_width": inSepalWidth,
                                 "petal_length": inPetalLength,
                                 "petal_width": inPetalWidth } } }
headers = {}
results = requests.post('http://127.0.0.1:9090/rpc'
                        data=json.dumps(payload),
                        headers=headers)
if not results.ok:
    <handle error>

result_list = results.json()['result']
...
```

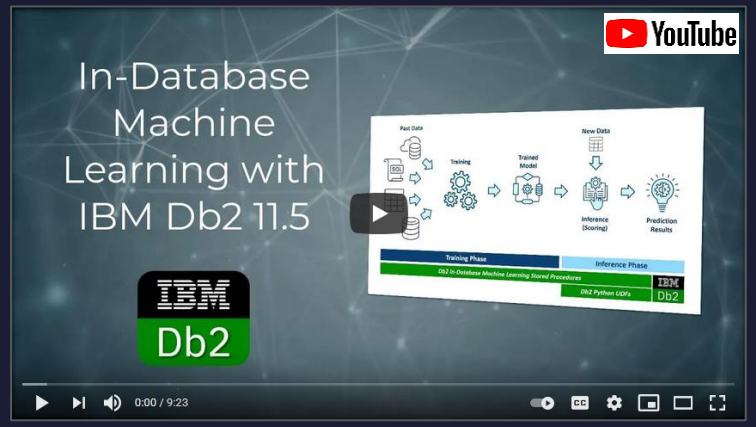
Db2 Python UDF code

Build payload/input to model's REST API

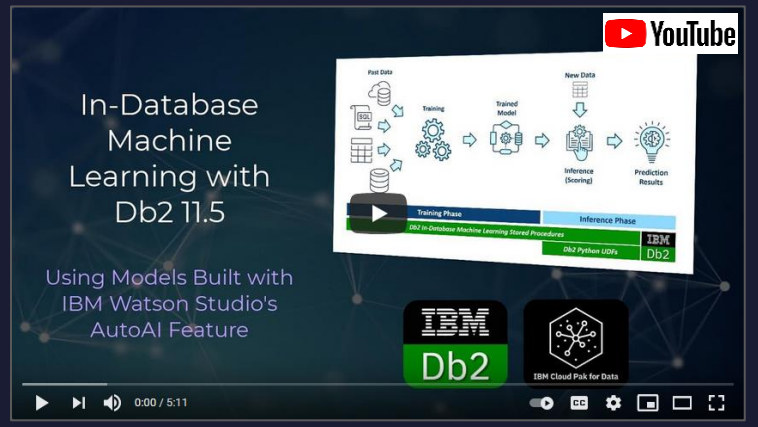
Post to API and perform error handling

Process results and determine value to return

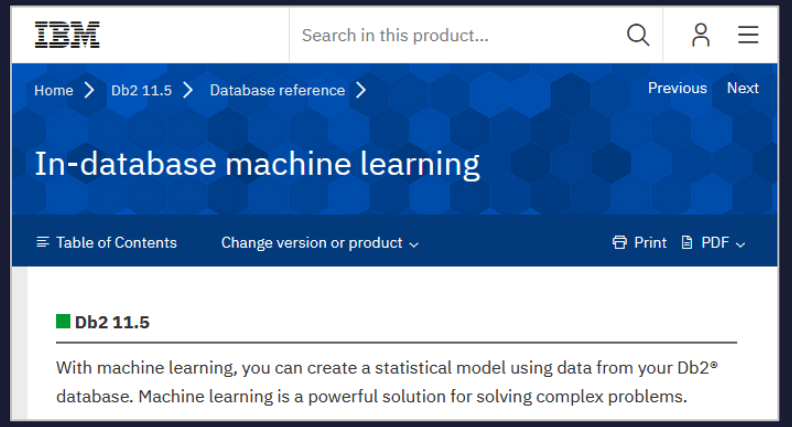
Additional Resources



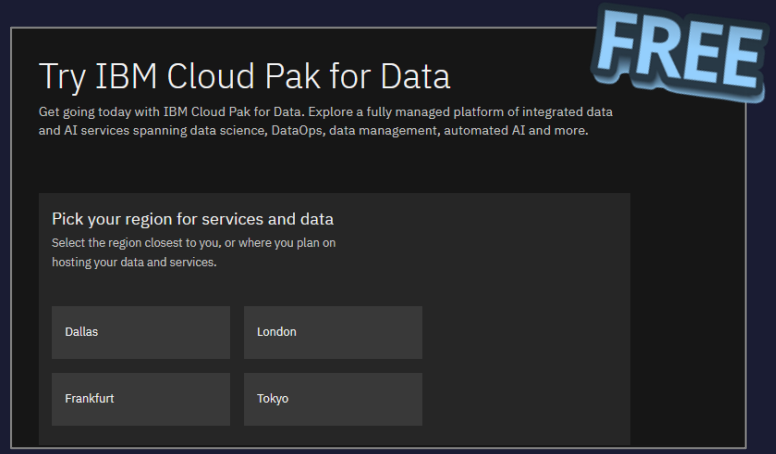
https://youtu.be/AEMrmS7-rZM



https://youtu.be/nYKR6WUOgBA



https://www.ibm.com/docs/en/db2/11.5?topic=content-in-database-machine-learning



https://dataplatform.cloud.ibm.com/registration/stepone
https://cloud.ibm.com/catalog/services/watson-studio

My Sample Code on GitHub

https://github.com/kschlamb/db2_machine_learning

IBM Db2

Other Samples

https://github.com/IBM/db2-samples/tree/master/In_Db2_Machine_Learning