Using DB2 LUW with DSX (Data Science Experience) to solve practical machine learning problems

Mark Ryan

IBM Canada

Session code: G02

Mon, Apr 30, 2018 (02:00 PM - 03:00 PM)

Db2 LUW

Agenda

- Introduce Data Science Experience (DSX)
- Show steps to set up Db2 as a data source for DSX
- Show steps to complete a working example in DSX that applies deep learning to solve a practical problem
- Next steps & resources

Who will find this session useful

- Familiar with Db2
- Not a machine learning guru
- Wants to take advantage of DSX / Watson Studio
- Looking for a straightforward way to apply machine learning techniques to everyday operational problems involving Db2 family data sources

Why machine learning matters

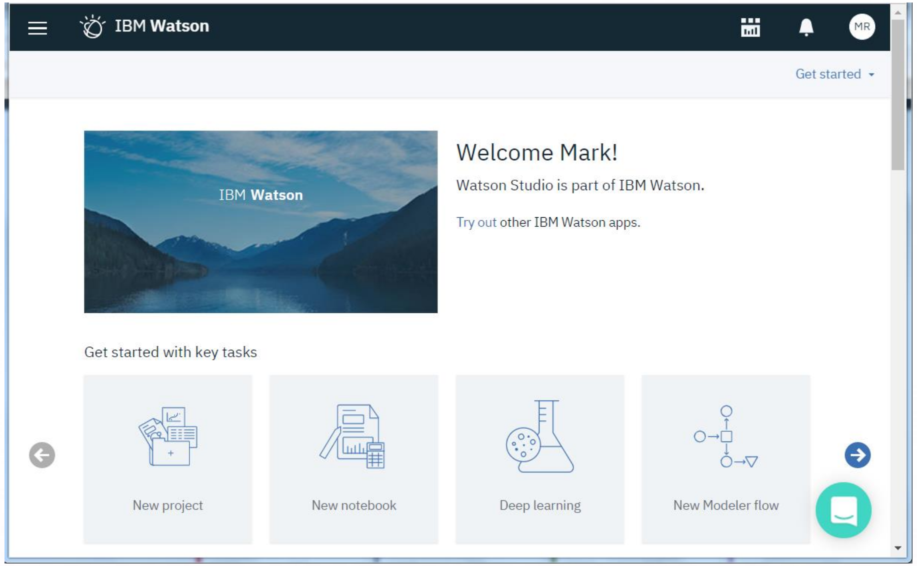
- Artificial intelligence is yielding results
- A shift is happening "from programming computers to showing computers"
- ML is becoming to our industry what calculus is to mathematics
- Good news: data analysis is the key prerequisite for ML

Why DSX matters

- Machine learning is undergoing a democratization
- DSX provides a self-contained, end-to-end, accessible way to apply machine learning
- You don‘t need to be a Data Scientist to get a lot out of DSX!
- Many data sources available to tap into via DSX, including Db2
- The public web instantiation of DSX is now part of Watson Studio

What is Data Science Experience (DSX)?

- DSX / Watson Studio includes tutorials, tools, articles, data sets and working code examples in the form of notebooks
- These capabilities also available in:
  - DSX Local
  - DSX Desktop
  - Integrated Analytics System
  - IBM Cloud Private for Data



What are notebooks?

- Notebooks are working code examples that incorporate both human-readable documentation and executable code (e.g. Scala, Python)
- Sample Notebooks in DSX range from end-to-end scenarios (e.g. classifying tumours) to specific actions (e.g. access Db2 from Scala)

Apply linear regression on the prepared data

Apply multiple linear regression on the prepared data to get the prediction model.

```
# Linear Multiple Regression
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(v10X, v10Y)
print("Coef: " + str(regressor.coef_))
print("Intercept: " + str(regressor.intercept_))
```

```
Coef: [ 0.00878584  0.02903981 -0.00335873]
Intercept: 0.0604897840556
```

Steps to set up Db2 as a data source for DSX

- Create required items:

Object	Where created	Credentials
Secure Gateway	IBM Cloud	Local host & port (for definition of the Destination with the Secure Gateway)
Client	Db2 system	NA
API call in notebook to access Db2	DSX	Local ID & password; Cloud host & port from Secure Gateway Destination properties

- Update Python code in notebook to refer to the data source with the appropriate credentials

Steps to set up a Db2 data source for DSX – (1|3)

- Create Secure Gateway – details here; summary of steps below:
  1. Log into IBM Cloud – from Dashboard select Create Service
  2. Search on Secure Gateway and select Free for pricing plan and Create
  3. Follow steps to create a Client
  4. Follow steps to create a Destination using the credentials for your Db2 system
  5. You will get the following credentials needed in subsequent steps:
     - Gateway ID
     - Cloud host
     - Cloud port

Steps to set up a Db2 data source for DSX – (2|3)

- Set up Client on Db2 system
  1. Log into IBM Cloud – from Secure Gateway download appropriate client (Docker a good bet)
  2. Install Client on Db2 system
  3. In Docker environment:
     - start client using Gateway ID from Secure Gateway Settings:

       ```
       Docker run -it ibmcom/secure-gateway-client pqDl217spyq_prod_ng
       ```

     - run acl command acl allow host:port for Db2 system:

       ```
       acl allow 10.0.9.93:3700
       ```

Steps to set up a Db2 data source for DSX – (3|3)

- Update notebook in DSX to tie everything together:
  1. Specify correct credentials:
     - Cloud host & port from Secure Gateway Destination Object settings
     - User and password from Db2 system
  2. You have a several options for the Python object to connect with your Db2 table – example below uses IdaDatabase object

```
dsn_uid = "";  # e.g.  db104434
dsn_pwd = ""    # e.g. xxxx
dsn_hostname =".integration.ibmcloud.com"  # e.g.  awh-yp-small03.services.dal.bluemix.net
dsn_port = "16833"   # e.g.  50001
dsn_database = "SAMPLE"   # e.g. BLUDB
```

Create the database connection

The following code snippet creates a connection string `connection_string` and uses the `connection_string` to create a Db2 connection object

```
connection_string='jdbc:db2://'+dsn_hostname+':'+dsn_port+'/'+dsn_database+':user='+dsn_uid+';password='+dsn_pwd+";"
idadb=IdaDataBase(dsn=connection_string)
```

Using Deep Learning in DSX to solve a practical problem

- A key aspect of a Db2 ticket is **Time to relief** (TTR) – the time taken to provide the customer with a way to address their primary symptom
- It would be very helpful to be able to predict as soon as a ticket is opened whether TTR will be more or less than 24 hours
- Goal: apply deep learning in DSX to predict when a ticket is first opened whether TTR will be greater than or less than 24 hours

Why this example is relevant

- Deep learning is the machine learning approach that is currently producing the most ground-breaking results
- Traditionally, deep learning has been applied mostly to unstructured data (e.g. Images, audio)
- This example is a simple illustration of:
  - Relatively new approaches to applying deep learning to structured data
  - How to apply deep learning to data in Db2 using DSX

Steps in this project (1 of 4)

- Research approaches to deep learning with structured data:
  - Great article on structured deep learning
  - Kaggle competition with great examples of applying DL to structured data
  - Fantastic course covering examples of deep learning on structured data
- Select the framework:
  - Notebooks in DSX as a development environment
  - Keras as a deep learning framework:
    - Open
    - Flexible enough for this application without being overwhelmingly complex (contrast with Tensor Flow, the framework on which Keras abstracts)
    - Widely used – large community; many questions already answered

Steps in this project (2 of 4)

3. Select features (columns) to feed into the model:
   - Pick only from subset of data that is available when the ticket is first opened
   - Avoid dependent features
   - Selected features covered three classes of data:

Continuous	Categorical	Text
Age	Country	Abstract
Hours	Analyst	Resolution

Steps in this project (3 of 4)

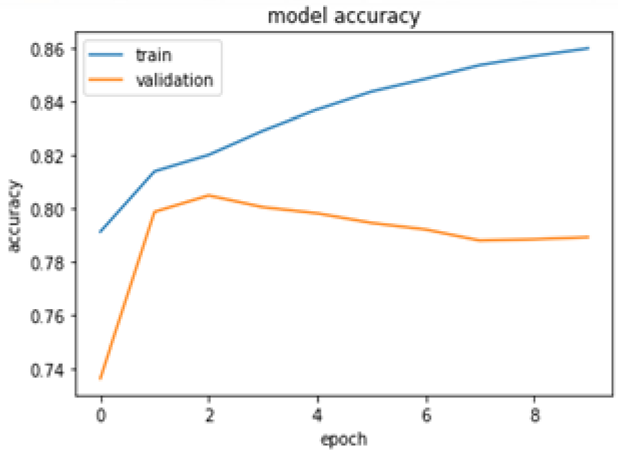
- Iteratively run model:
  - Tune hyperparameters (e.g. Learning rate, parameters to control overfitting, batch size)
  - Experiment with model structure (e.g. Add additional dense layers prior to output, try different optimization and activation functions)
- Goal: best possible validation accuracy:
  - Accuracy of the model‘s prediction of TTR <> 24 hours vs. Actual TTR for the validation set

Steps in this project (4 of 4)

- Run experiments on input features:
  - Experiment with subsets of the available data by feature class:
    - Text feature only (Abstract)
    - Categorical/continuous features only
    - Combination of all three classes of features
  - Experiment with calculated features:
    - Include time since GA of release as a calculated continuous feature
    - Exclude zero-day TTR
    - Distort ticket severity feature to highlight extremes (Sev 1 and Sev 4)
    - Include day of week, month, year, day as calculated continuous features
  - Experiment with scope of the input data set:
    - Db2 only
    - Db2 + other Analytics products

Lessons learned

1. Keras in DSX a solid deep learning platform
2. Tuning hyperparameters can be a distraction:
   - Fundamental problems with accuracy were solved by correcting input data & model problems
   - Tuning hyperparameters helped stability but not accuracy
3. Avoiding overfitting is key
4. More data = better accuracy



Demo

```
BATCH_SIZE = 2000
epochs = 20
print("LR ",learning_rate)
print("dropout ",dropout_rate)
print("L2 lambda ",l2_lambda)
print("batch size ",BATCH_SIZE)
print("epochs ",epochs)


model = get_model()
modelfit = model.fit(X_train, dtrain.target, epochs=epochs, batch_size=BATCH_SIZE
          , validation_data=(X_valid, dvalid.target), verbose=1)
```

```
LR  0.001
dropout  0.003
L2 lambda  7.5
batch size  2000
epochs  20
Train on 105230 samples, validate on 26308 samples
Epoch 1/20
105230/105230 [==============================] - 60s - loss: 269.3049 - acc: 0.7026 - val_loss: 184.2830 - val_acc: 0
.6529
Epoch 2/20
105230/105230 [==============================] - 54s - loss: 131.7222 - acc: 0.7649 - val_loss: 87.9015 - val_acc: 0.
6950
Epoch 3/20
105230/105230 [==============================] - 54s - loss: 61.5127 - acc: 0.7797 - val_loss: 39.9705 - val_acc: 0.6
809
Epoch 4/20
105230/105230 [==============================] - 54s - loss: 27.2616 - acc: 0.7904 - val_loss: 17.2707 - val_acc: 0.6
927
```

Next steps

- Apply the same framework to other problems:
  - Predicting tickets with low NPS
  - Predicting code changes that are likely to cause regressions
  - Predicting tickets that are likely to require code changes to resolve
  - Predicting tickets that are likely to result in Duty Manager calls
- Deployment of models in production

Resources

- Andrew Ng intro to ML
- Andrew Ng Deep Learning Curriculum
- Jeremy Howard Deep Learning
- Competitions on Kaggle
- Code for the example in this presentation
- Fantastic end-to-end example of using DSX with Db2 Warehouse on Cloud
- Essential source for ideas & inspiration: Towards Data Science

Mark Ryan
IBM
mryan@ca.ibm.com

Session code: G02

Please fill out your session evaluation before leaving!