Machine Learning with Qlik®

Qonnections Hands-on Workshop

TABLE OF CONTENTS

INTRODUCTION

In these hands-on exercises we will be using a Python Server-Side Extension (SSE) to demonstrate machine learning capabilities in Qlik.

In the first exercise we will generate a timeseries forecast. The goal here is to simply understand the communication flow between Qlik and the SSE, and be familiar with a reusable capability that is relevant for many datasets.

In the next exercise we will prepare data for supervised machine learning, train predictive models using different algorithms and then test and evaluate these models. All of this will be done through function calls to the SSE from a Qlik Sense app.

The final exercise will make use of the trained model to make predictions in real-time using a Qlik Sense app.

BEFORE YOU BEGIN

1. Resume the Virtual Machine for this workshop
2. Navigate to C:\qlik-py-tools\
3. Right click Qlik-Py-Start.bat and select Run as Administrator

In the instructions below, **blue** represents things you need to do, commentary in **grey** provides context and **plum** represents code, keywords or things you should generally not ignore.

EXERCISE 1: FORECASTING

The SSE provides forecasting capabilities using Prophet; an open source library published by the Data Science team at Facebook.

Objectives

- Learn to use SSE functions through Qlik expressions
- Understand the communication flow between Qlik and the SSE
- Review project assets and documentation available to you

Open the app

Open the "Qonnections – Forecasting" app and jump into the first sheet: "Hands-on Exercise".

The data represents emergency department attendances for hospitals in Victoria, Australia.

Add a new measure to the line chart

Go into Edit mode, add a new measure to the line chart and go into the Expression Editor.

Type the following expression:

```
PyTools.
```

This is the name of the analytic connection to the SSE as set up in QMC.

You should see the capabilities made available by the SSE. This is a list of reusable functions that provide data science and machine learning capabilities to Qlik.

Use the forecasting function

We will use the Prophet function to generate the forecast.

Type or copy in the following expression:

```
Pytools.Prophet([Month Start], Sum(Attendances), 'freq=M')
```

The inputs to the function are a time dimension, the measure to be forecasted, and additional arguments to control the behavior of the algorithm. In this case we pass freq=M as the data is being provided at a monthly frequency.

Apply the changes and observe that a forecast is now available on the line chart.

Improve the forecast

The output of the function can be controlled by passing arguments. For this function all arguments are passed through a single string, with key words separated by commas.

Update the forecast expression as below:

```
Pytools.Prophet([Month Start], Sum(Attendances), 'freq=M, take_log=true, debug=true')
```

By adding these arguments we're taking a logarithm of the input values before applying the forecasting algorithm. We also turned on debugging which will help us in the next step.

You may notice that taking a log has improved the forecast. This is a technique used to improve the stationarity of the timeseries. You may not know what all that means, but such things are usually discovered by digging a bit deeper into the data science behind these capabilities.

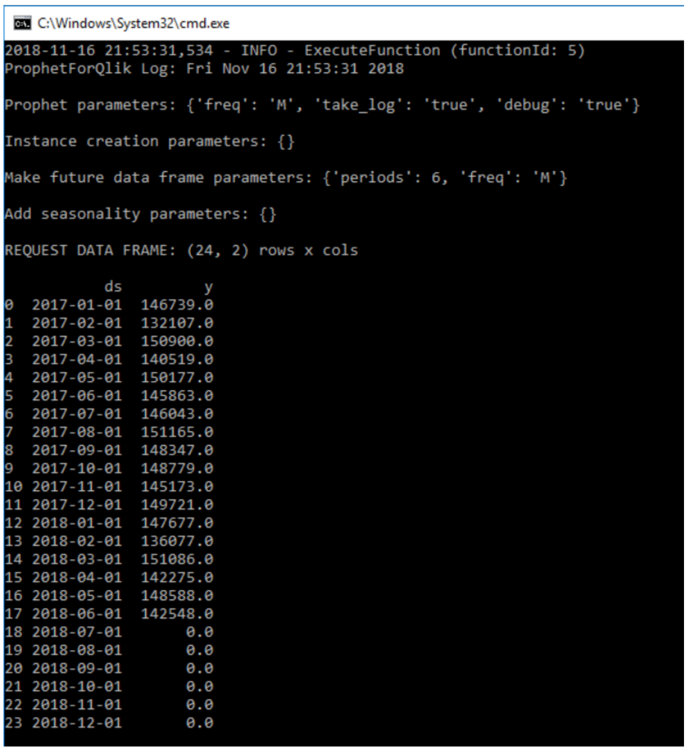
There are other arguments that can be used to influence the results. You'll find a link to the project documentation towards the end of this exercise.

Review communication between Qlik and the SSE

Turning on debugging in our last expression allows us to review the communication flow between Qlik and the SSE.

Go to the terminal being used to run the SSE, and scroll through the output following the last 'INFO - ExecuteFunction' message.

Alternatively, you can also look at the log "Prophet Log 1.txt" found under C:\qlik-py-tools\qlik-py-env\core\logs on the VM.



```
2018-11-16 21:53:31,534 - INFO - ExecuteFunction (functionId: 5)
ProphetForQlik Log: Fri Nov 16 21:53:31 2018

Prophet parameters: {'freq': 'M', 'take_log': 'true', 'debug': 'true'}

Instance creation parameters: {}

Make future data frame parameters: {'periods': 6, 'freq': 'M'}

Add seasonality parameters: {}

REQUEST DATA FRAME: (24, 2) rows x cols

            ds         y
0   2017-01-01  146739.0
1   2017-02-01  132107.0
2   2017-03-01  150900.0
3   2017-04-01  140519.0
4   2017-05-01  150177.0
5   2017-06-01  145863.0
6   2017-07-01  146043.0
7   2017-08-01  151165.0
8   2017-09-01  148347.0
9   2017-10-01  148779.0
10  2017-11-01  145173.0
11  2017-12-01  149721.0
12  2018-01-01  147677.0
13  2018-02-01  136077.0
14  2018-03-01  151086.0
15  2018-04-01  142275.0
16  2018-05-01  148588.0
17  2018-06-01  142548.0
18  2018-07-01       0.0
19  2018-08-01       0.0
20  2018-09-01       0.0
21  2018-10-01       0.0
22  2018-11-01       0.0
23  2018-12-01       0.0
```

Observe the following:

- The input data from Qlik contains the future periods with NULL values passed for the measure. These future values for the time dimension need to be prepared in Qlik.
- The y values in the input data frame have been transformed due to the take_log=true argument.
- The sample results show that there are several columns available for output, including the trend component of the timeseries, as well as upper and lower limits.
- The forecast is returned for all periods, so we can compare actual values for historical data against predictions from our forecasting model. To only return the forecast for future periods you can include the following in your argument string: ‘return=y_then_yhat’.

Explore the second sheet in the app

Now, hopefully, with a better sense of what’s going on, head back to the app and to the “Emergency Attendances Forecast” sheet.

Here we have added upper and lower limits to the forecast. We have also added a variable slider that controls an argument passed to the functions. Review the expressions for these measures under Master Items.

In view mode, make selections for different hospitals and observe how the forecast and limits are generated on the fly.

Try adjusting how well the trend fits to historical data using the slider. How does this affect upper and lower confidence intervals?

The slider controls how well the forecast fits to historical data. Trying to fit a model too closely to historical data results in ‘overfitting’, which can lead to poor predictions and high uncertainty for new data. This is a problem that we need to watch out for in machine learning.

Know where to find more information

The functions you’ve used are part of an open source project listed on GitHub and Qlik Branch.
https://github.com/nabeel-oz/qlik-py-tools
https://developer.qlik.com/garden/5af5217ab2606a3c2c1f4d1d

The open source project includes sample apps and help on using Prophet’s capabilities in Qlik.
https://github.com/nabeel-oz/qlik-py-tools#usage

Refer to the Forecasting documentation under the Usage section to explore the full capabilities of the Prophet functions and how to apply them to your own data.

EXERCISE 2: SUPERVISED MACHINE LEARNING

In this exercise we will go through the complete flow of training and testing a supervised machine learning (ML) model. The functions we will use are implemented using the popular Scikit-Learn library.

Objectives

- Understand the steps involved in machine learning
- Train a model using different estimators
- Evaluate the model using results from testing
- Appreciate how we can improve the model

Review the problem

Open the “Qonnections – ML Model” app. Open the Data Manager and review the table in the Associations view.

We will attempt to train a model that can predict whether a person is likely to miss their medical appointment. The Train-Test table presents the data we will use to train and then test the model.

The target is the No-show column, which has two classes; “Yes” and “No”. This is considered a classification problem as the target has discrete classes. By contrast, a regression problem involves predicting a continuous variable such as sales revenue.

We have held back some data from this app so we can simulate predictions on unseen data in the next exercise.

Add estimators for training the model

We’ll choose a few classification algorithms to be used for training and evaluation.

Click on Add Data and then Manual Entry. Name the table “**Estimators**”.

Copy the rows below, click the first column header in Qlik and hit Ctrl+V to add the table below.

Model Name	Estimator	Hyperparameters
NoShow-RF	**RandomForestClassifier**	**n_estimators=10|int, criterion=gini|str**
NoShow-LR	**LogisticRegression**	

If you’re confident that you’ll have enough time, add the third estimator below. If you’re not sure, just stick to the two above as each estimator will increase our load time by about 3 minutes.

NoShow-SGD	**SGDClassifier**	**loss=modified_huber|str**

Click Add Data.

The Model Name is a unique name assigned to the model. This will be used for persisting the model on disk and later calling it for various functions.

The Estimator field can be populated with any valid class from the scikit-learn library. In this case any classification algorithm can be selected, however in the interest of time we will just select three.

Hyperparameters are parameters that will not be learnt during training. Each scikit-learn class has its own set of parameters. The SSE will simply parse the string passed through this field, convert the arguments to the correct data type in Python and then pass them on to the algorithm.

Review the load script

Navigate to the Data Load Editor.

Note that several sections have been prepared beforehand, but are not being executed at the moment due to the “Exit Script” section. Glance through these sections and review the comments to get a sense of how the data is prepared for ML, trained and then tested.

Train and test the model

Switch to the VM and open the following folder:

C:\qlik-py-tools\qlik-py-env

Back in the app, move the “Exit Script” section to the end and run the data load. This may take around 7 to 9 minutes.

Note that the models are saved to disk in the “models” subfolder in the directory that you have opened in the VM.

Evaluate the model

Open the “Model Evaluation” sheet and review the score for each algorithm. By default, this score represents the accuracy of the model.

You should have a decent accuracy of around 78% for the best model. Try to understand these results better using the Confusion Matrix which shows predicted versus actual labels.

Navigate to the “Key Drivers” sheet and observe which features had the most influence for each algorithm.

Improve the model

We will now try to improve on these results. You may think that this involves choosing better algorithms or tuning the hyperparameters passed to the algorithms. These things do play a part in improving results, but by far the most important component is the features that you are feeding to the model.

Add a new section to your load script after the “Suburbs & Regions” section. We will calculate how often an individual has previously skipped an appointment, and whether the person has several appointments on the same day

Copy and paste the script below into the new section:

What is feature engineering?

Feature engineering is fundamental to machine learning. It is the process of preparing dimensions and measures that can help explain the target.

This is something that Qlik is naturally good at; bringing data sources together, deriving dimensions and measures and consolidating into the single table required for machine learning.

This process has traditionally been done programmatically by Data Scientists, however, auto feature engineering tools are gaining popularity.

```
// Count previous no-shows for each patient as at AppointmentDay

TempNoShows:
LOAD
    PatientId,
    AppointmentDay,
    AppointmentID,
    if(PatientId = Peek(PatientId),
        if([No-show] = 'Yes', 1, 0) + Peek([Previous No-shows]),
        if([No-show] = 'Yes', 1, 0)) as [Previous No-shows];
LOAD DISTINCT
    PatientId,
    AppointmentDay,
    AppointmentID,
    [No-show]
RESIDENT [Train-Test]
Order by PatientId, AppointmentDay, AppointmentID Asc;

LEFT JOIN([Train-Test])
LOAD
    PatientId,
    AppointmentDay,
    AppointmentID,
    [Previous No-shows]
RESIDENT TempNoShows;

Drop table TempNoShows;

// Calculate the number of appointments a patient has on the same day

TempSameDay:
LOAD DISTINCT
    PatientId,
    AppointmentDay,
    Count(DISTINCT AppointmentID) as AppointmentsSameDay
RESIDENT [Train-Test]
GROUP BY PatientId, AppointmentDay;

LEFT JOIN([Train-Test])
LOAD
    PatientId,
    AppointmentDay,
    AppointmentsSameDay
RESIDENT TempSameDay;

Drop table TempSameDay;
```

Now in the “Feature Preparation” section add these new fields to the “features” table:

```
[Previous No-shows] as f_prev_noshows,
[AppointmentsSameDay] as f_same_day_appts
```

You’ll also need to add the new fields to the “feature_definitions” table:

```
f_prev_noshows, feature, int, scaling
f_same_day_appts, feature, int, scaling
```

Finally, go to the “Model Training & Testing” section and add these fields to the “TEMP_TRAIN_TEST” table. Note that f_daystoappt is no longer the last column and needs to be updated accordingly.

```
f_daystoappt & '|' &
f_prev_noshows & '|' &
f_same_day_appts AS N_Features
```

Run the load and check whether we have improved the model using the “Model Evaluation” sheet.

Review the “Key Drivers” sheet and identify the most important features.

Know where to find more information

The GitHub open source project for this SSE includes sample apps and documentation. https://github.com/nabeel-oz/qlik-py-tools#usage

Refer to the Machine Learning documentation under the Usage section to explore the full capabilities.

Documentation on the scikit-learn library can be found here.

If you’re looking for data to try your hand at machine learning, Kaggle.com is a good place to start. The dataset used in this app can be found here.

EXERCISE 3: PREDICTIONS

In this exercise we will use the model trained in Exercise 2 to make predictions on unseen data.

Objectives

- Experience how a trained model can be used to make predictions
- Appreciate how Qlik's Associative Difference applies to these predictions

Explore the data

Open the "Qonnections – ML Predictions" app.

Explore the data using the "Appointment No-shows" sheet.

This app includes historical data for April and May, where we know if an appointment was a no-show. This data was used for training the mode in the previous exercise. We also have 26,451 records for June. This data was held back during the previous exercise.

Do any of the visualizations help you identify a large percentage of no-shows?

The most influential feature identified in Exercise 2 was the number of previous no-shows for the patient. Select patients with previous no-shows on the scatter plot and see how the percentage of no-shows compares with the overall average.

Add the prediction expressions

Navigate to the "Predicted No-shows" sheet and go into Edit mode.

We will add expressions to get predictions from the model trained in Exercise 2.

Select the table and note that the two columns at the end are displayed conditionally based on a variable controlled by the "Predict" and "Pause" buttons.

Add the following expression for the second-last column.

```
PyTools.sklearn_Predict('NoShow-RF', $(vFeaturesExpression))
```

We're calling the previously trained model "NoShow-RF" and passing in the features using a variable.

Open the variable panel and note that "vFeaturesExpression" is a script variable, populated with the expression needed to pass input features to the model. These features need to be named and ordered exactly as they were during model training, and this variable provides a convenient way of avoiding mistakes.

The "vFeaturesExpression" variable is populated by a call to the SSE in the "Feature Expression" section of the load script.

Add the following expression for the last column in the table.

```
PyTools.sklearn_Predict_Proba('NoShow-RF', $(vFeaturesExpression))
```

This column will provide predicted probabilities for each class.

Get predictions from the model

In the “Predicted No-shows” sheet, switch to View mode and hit the Predict button.

Select “Unknown” from the No-show column. Note how predictions are being called in real-time as you explore the data.

Select “Yes” from the Prediction column. This column was created as a dimension so the predictions received from the model are available for selections! This means you can now use these predictions to ask your next question.

We have successfully predicted appointments most likely to be missed; an insight that can help to improve the process of following up with patients and providing better service.