Published in Towards Data Science

Pol Ferrando Follow

Jan 22, 2019 · 11 min read · Listen

Introduction to BigQuery ML

A few months ago Google announced a new Google BigQuery feature called BigQuery ML, which is currently in Beta. It consists of a set of extensions of the SQL language that allows to create machine learning models, evaluate their predictive performance and make predictions for new data directly in BigQuery.

Source: https://twitter.com/sfeir/status/1039135212633042945

split, etc. In addition, it reduces the training time of models because it works directly where the data is stored (BigQuery) and, consequently, it is not necessary to export the data to other tools.

But not everything is an advantage. First of all, the implemented models are currently limited (although we will see that it offers some flexibility), which probably will always be the case due to the fact of adapting to SQL. Secondly (and, in my opinion, more important), even though BQML makes model training easier, a person who is not familiar with machine learning can still have difficulties interpreting the model they have created, evaluating its performance and trying to improve it.

In this post, I will explain the main functions of BQML and how to use them to create our model, evaluate it and use it to make predictions. This process will consist of the following steps:

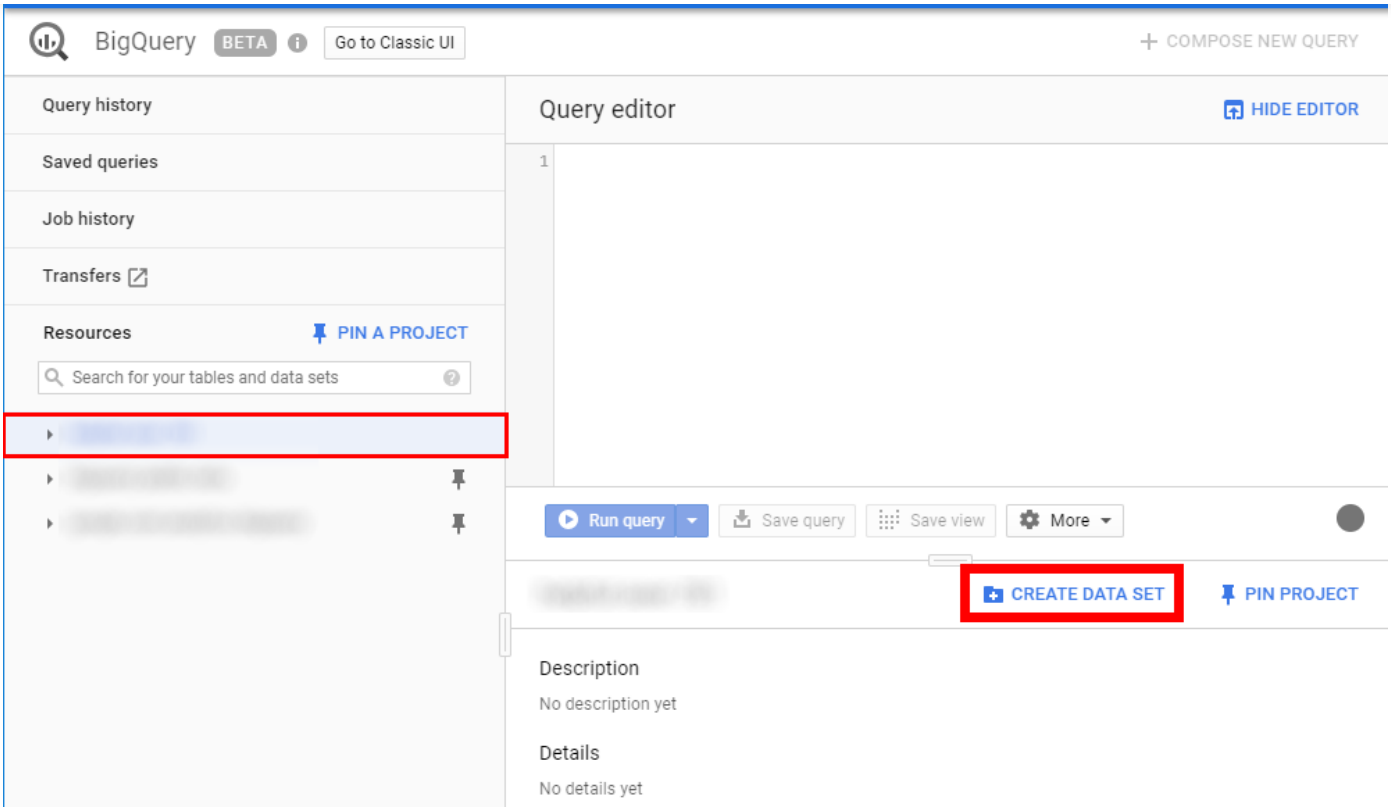
1. Create a dataset (optional)
2. Create a model
3. Model information (optional)
4. Evaluate the model
5. Predict

Create a dataset (optional)

As with BigQuery (BQ) tables, the model must be saved in a data set, so first you have to decide in which data set you want to save the model: an existing one or in a new one.

If your case is the latter, creating a new data set is as simple as:

1. In the BQ interface, select the project in which you want to create the dataset and

2. Name the new data set and choose a location where the data will be stored and the expiration. You can find more information about these fields in this link.

Create a model

In supervised machine learning, a training data set whose response variables are known is used to generate a model which captures the underlying patterns of the data so that it can predict the outcome of unseen data.

BQML allows you to do this process directly within BigQuery. Currently, it supports three types of models:

1. **Linear regression**. It is used to predict the result of a continuous numeric variable, such as income.

2. **Binary logistic regression**. It is used to predict the result of a categorical variable

3. **Multinomial logistic regression** (or multiclass). It is used to predict the result of a categorical variable with more than two classes.

To create (and train) a model with BQML, the following syntax has to be used:

```
#standardSQL
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL}
`project.dataset.model_name`
OPTIONS(model_option_list)
AS query_statement
```

This query will create a model (CREATE MODEL) with the specified options (OPTIONS) and using the result of a query (AS) as training data. We have to specify:

1)The name of the model and where it has to be saved. **CREATE MODEL** creates and trains the model (which will be saved with the name "**model_name**" inside the specified data set) as long as there is no model already created with the same name. If the model name exists, the query with CREATE MODEL will return an error. To avoid this error, we can use two alternatives:

- CREATE MODEL IF NOT EXISTS, which creates and trains the model only if there is no model already created with the same name.
- CREATE OR REPLACE MODEL, which creates the model (if it does not exist) or replaces it (if it exists) and trains it.

2) **model_option_list** . A list specifying some options related to the model and the training process. The format is as follows: *option1 = value1, option2 = value2,* … The two most important options are:

- model_type (mandatory): specifies the type of model we want to train: *linear_reg* for linear regression or *logistic_reg* for binary or multiclass logistic regression.
- input_label_cols: specifies the column name of the table with the training data that

Although BigQuery ML has default options for model training, it offers some flexibility to choose options related to avoiding overfitting and the optimization of model parameters. For example, we can apply regularization L1 or L2, split the data in a training set and a validation set, or set the maximum number of iterations of the gradient descent. You can find all the available options in this link.

3) **query_statement**. Query that generates the table that will be used as training data. One of the advantages of BigQuery ML is that it is responsible for data transformation for model training. In particular:

- Categorical features (of type BOOL, STRING, BYTES, DATE, DATETIME or TIME) are one-hot enconded (i.e., converted into a binary variable for each class). Due to the problem known as *multicollinearity*, this is not recommended if you want to draw conclusions about the relationships between the features and the response variable.
- Numerical features (type NUMERIC, FLOAT64 or INT64) are standardized for both training data and future predictions.
- NULL values are replaced by the average in the case of numerical variables or by a new class that groups all these missing data in the case of categorical features.

Regarding the response variable, it must be taken into account that:

- In linear regression can not have infinite or NaN values.
- In binary logistic regression it has to have exactly two possible values.
- In multiclass logistic regression you can have a maximum of 50 different categories.

For example, let's imagine that we want to predict whether a web session will end up buying or not depending on several features related to the user's browsing activity (number of page views, session duration, type of user, the device he uses and whether it is paid traffic or not). In case you want to follow this example, we will use the Google Analytics test data set offered by BigQuery. To create the model, we would use the


```
OPTIONS(model_type='logistic_reg',
       input_label_cols=['isBuyer'])
AS
SELECT
 IF(totals.transactions IS NULL, 0, 1) AS isBuyer,
 IFNULL(totals.pageviews, 0) AS pageviews,
 IFNULL(totals.timeOnSite, 0) AS timeOnSite,
 IFNULL(totals.newVisits, 0) AS isNewVisit,
 IF(device.deviceCategory = 'mobile', 1, 0) AS isMobile,
 IF(device.deviceCategory = 'desktop', 1, 0) AS isDesktop,
 IF(trafficSource.medium in ('affiliate', 'cpc', 'cpm'), 1, 0) AS
isPaidTraffic
FROM
 `bigquery-public-data.google_analytics_sample.ga_sessions_*`
WHERE
 _TABLE_SUFFIX BETWEEN '20160801' AND '20170630'
```

Since our response variable is categorical with two classes (1 = “with purchase” or 0 = “without purchase”), we’ve had to specify in the options that the type of model is a logistic regression (logistic_reg). Also, note that the response variable is called “isBuyer”, so we’ve had to specify that in the options too.

Model information (optional)

In linear models, each explanatory variable has an associated coefficient (or weight) that determines the relationship between this feature and the response variable. The greater its magnitude, the greater impact it has on the response variable. Furthermore, the positive (negative) sign indicates whether the response increases (decreases) when the value of this explanatory variable increases (or, in the case of categorical variables, the category is present).

In BigQuery ML, we can get the weights of the trained model with the following query:

```
#standardSQL
```


As mentioned before, if you don't convert the categorical variables into binary "manually" in your query (as we have done with isMobile and isDesktop, for instance), each possible category will have a weight and the reliability of the coefficients will be compromised due to multicollinearity.

For example, the model we created in the previous section has the following coefficients:

Job information | Results | JSON | Execution details

Row	processed_input	weight	category_weights.category	category_weights.weight
1	pageviews	0.0880949403447221		
2	timeOnSite	2.989341692142403E-4		
3	isNewVisit	-1.0179891084233867		
4	isMobile	-0.2943824436397999		
5	isDesktop	0.405665230120851		
6	isPaidTraffic	-0.3117957621457784		
7	__INTERCEPT__	-4.628392114215315		

That is, being a session of a new user, using a mobile device or accessing the web through a paid channel decreases the probability of that visit ending in purchase. While using desktop or spending more time on the site increases the probability of converting.

Evaluate the model

Once we have the trained model, we need to assess its predictive performance. This always has to be done on a test set different from the training set to avoid overfitting, which occurs when our model memorizes the patterns of our training data and consequently it is very precise in our training set but it is not capable of making good predictions in new data.

iteration. The expected result is that the loss decreases in both sets (ideally, to 0, which means that the model is always right).

- ML.EVALUATE. Provides the most common metrics to assess the predictive performance of a model. This function can be used for any type of model (linear regression, binary logistic regression, multiclass logistic regression), but the metrics will be different depending on whether it is a regression or a classification task.
- ML.CONFUSION_MATRIX. Returns the confusion matrix for a given data set, which allows us to know the correct predictions and the errors for each possible class in a classification model. It can only be used for classification models, that is, logistic regression and multiclass logistic regression.
- ML.ROC_CURVE. This function allows us to construct the ROC curve, which is a graphical visualization used to evaluate the predictive ability of a binary classification model. In this case, it can only be used for binary logistic regression models.

In this post we will focus on ML.EVALUATE, but we will give the syntax and examples for the other functions in case someone is interested in using them.

ML.EVALUATE

To evaluate a previously created model, the following syntax has to be used:

```
#standardSQL
SELECT *
FROM ML.EVALUATE(MODEL `project.dataset.model_name`,
          {TABLE table_name | (query_statement)}
          [, STRUCT(XX AS threshold)])
```

Where we have to specify:

- The model.
- The table for which we want to compute the evaluation metrics, which can be the

values). If no table or query is specified, it will use the validation set (if specified when creating the model) or the training set (if a validation set was not specified).

- In the case of a logistic regression, a threshold. This value is optional, and it specifies the value from which the predictions of our model (which are values between 0 and 1 that can be interpreted as probabilities that this observation is of class 1) will be for class 0 or for the class 1. By default, the threshold will be 0.5.

The result of this query is a single row with the most common metrics to evaluate the predictions of a model, which will depend on the type of model used.

In particular, the metrics that BigQuery ML provides for logistic regression and multiclass logistic regression models are:

- precision
- recall
- accuracy
- f1_score
- log_loss
- roc_auc

In the case of linear regression, they are:

- mean_absolute_error
- mean_squared_error
- mean_squared_log_error
- median_absolute_error
- r2_score


```
#standardSQL
SELECT *
FROM ML.EVALUATE(MODEL `project.dataset.sample_model`,
   (
     SELECT
        IF(totals.transactions IS NULL, 0, 1) AS isBuyer,
        IFNULL(totals.pageviews, 0) AS pageviews,
        IFNULL(totals.timeOnSite, 0) AS timeOnSite,
        IFNULL(totals.newVisits, 0) AS isNewVisit,
        IF(device.deviceCategory = 'mobile', 1, 0) AS isMobile,
        IF(device.deviceCategory = 'desktop', 1, 0) AS isDesktop,
        IF(trafficSource.medium in ('affiliate', 'cpc', 'cpm'), 1, 0)
AS isPaidTraffic
     FROM
        `bigquery-public-data.google_analytics_sample.ga_sessions_*`
     WHERE
        _TABLE_SUFFIX BETWEEN '20170701' AND '20170801'
   ),
   STRUCT(0.5 AS threshold)
   )
```

Note that the dates used to generate the data are different from those used to create the model. The result of the previous query is:

Job information | Results | JSON | Execution details

Row	precision	recall	accuracy	f1_score	log_loss	roc_auc
1	0.4430379746835443	0.09776536312849161	0.9851952452667814	0.16018306636155608	0.049046802468474655	0.975576

Other functions to evaluate models

1. ML.TRAINING_INFO

Syntax:

```
#standardSQL
SELECT *
FROM ML.TRAINING_INFO(MODEL `project.dataset.model_name`)
```

```
#standardSQL
SELECT *
FROM ML.TRAINING_INFO(MODEL `project.dataset.sample_model`)
```

2. ML.CONFUSION_MATRIX

Syntax:

```
#standardSQL
ML.CONFUSION_MATRIX(MODEL `project.dataset.model_name`,
          {TABLE table_name | (query_statement)}
          [, STRUCT(XX AS threshold)])
```

Example:

```
#standardSQL
SELECT *
FROM ML.CONFUSION_MATRIX(MODEL `project.dataset.sample_model`,
   (
     SELECT
        IF(totals.transactions IS NULL, 0, 1) AS isBuyer,
        IFNULL(totals.pageviews, 0) AS pageviews,
        IFNULL(totals.timeOnSite, 0) AS timeOnSite,
        IFNULL(totals.newVisits, 0) AS isNewVisit,
        IF(device.deviceCategory = 'mobile', 1, 0) AS isMobile,
        IF(device.deviceCategory = 'desktop', 1, 0) AS isDesktop,
        IF(trafficSource.medium in ('affiliate', 'cpc', 'cpm'), 1, 0)
AS isPaidTraffic
     FROM
        `bigquery-public-data.google_analytics_sample.ga_sessions_*`
     WHERE
        _TABLE_SUFFIX BETWEEN '20170701' AND '20170801'
   ),
   STRUCT(0.5 AS threshold)
   )
```

```
#standardSQL
ML.ROC_CURVE(MODEL `project.dataset.model_name`,
           {TABLE table_name | (query_statement)},
           [GENERATE_ARRAY(thresholds)])
```

Example:

```
#standardSQL
SELECT *
FROM ML.ROC_CURVE(MODEL `project.dataset.sample_model`,
   (
     SELECT
        IF(totals.transactions IS NULL, 0, 1) AS isBuyer,
        IFNULL(totals.pageviews, 0) AS pageviews,
        IFNULL(totals.timeOnSite, 0) AS timeOnSite,
        IFNULL(totals.newVisits, 0) AS isNewVisit,
        IF(device.deviceCategory = 'mobile', 1, 0) AS isMobile,
        IF(device.deviceCategory = 'desktop', 1, 0) AS isDesktop,
        IF(trafficSource.medium in ('affiliate', 'cpc', 'cpm'), 1, 0)
AS isPaidTraffic
     FROM
        `bigquery-public-data.google_analytics_sample.ga_sessions_*`
     WHERE
        _TABLE_SUFFIX BETWEEN '20170701' AND '20170801'
   ),
   GENERATE_ARRAY(0.0, 1.0, 0.01)
   )
```

Predict

To use a model created with BigQuery ML to make predictions, the following syntax has to be used:

```
#standardSQL
```


This query will use a model (MODEL) and will make predictions of a new data set (TABLE). Obviously, the table must have the same columns as the training data, although it isn't necessary to include the response variable (since we do not need it to make predictions of new data). In logistic regression, you can optionally specify a threshold that defines from which estimated probability is considered as a final prediction one class or another.

The result of this query will have as many rows as the data set we have provided and it will include both the input table and the model predictions. In the case of logistic regression models (binary or multiclass), in addition to the class that predicts the model, the estimated probability of each of the possible classes is also provided.

And continuing with our example:

```
#standardSQL
SELECT *
FROM ML.PREDICT(MODEL `project.dataset.sample_model`,
   (
     SELECT
        IFNULL(totals.pageviews, 0) AS pageviews,
        IFNULL(totals.timeOnSite, 0) AS timeOnSite,
        IFNULL(totals.newVisits, 0) AS isNewVisit,
        IF(device.deviceCategory = 'mobile', 1, 0) AS isMobile,
        IF(device.deviceCategory = 'desktop', 1, 0) AS isDesktop,
        IF(trafficSource.medium in ('affiliate', 'cpc', 'cpm'), 1, 0)
AS isPaidTraffic
     FROM
        `bigquery-public-data.google_analytics_sample.ga_sessions_*`
     WHERE
        _TABLE_SUFFIX BETWEEN '20170701' AND '20170801'
   )
   )
```

Note that the column with the response (isBuyer) is not required. The result of the previous query is:

Job information Results JSON Execution details

Row	predicted_isBuyer	predicted_isBuyer_probs.label	predicted_isBuyer_probs.prob	pageviews	timeOnSite	isNewVisit	isMobile	isDesktop	isPaidTraffic
1	0	1	0.00788654273338147	1	0	0	1	0	0
		0	0.9921134572666185						
2	0	1	0.015756320159654655	1	0	0	0	1	0
		0	0.9842436798403453						
3	0	1	0.00788654273338147	1	0	0	1	0	0
		0	0.9921134572666185						
4	0	1	0.005750959250768079	1	0	1	0	1	0
		0	0.9942490407492319						

The first column returns the class that our model predicts for each new observation. The second and third columns give us the estimated probability for each of the classes (the class whose estimated probability is greater is the one in the first column). The rest of the columns are the data whose predictions we have requested.