Basic Feature Engineering in BQML

Learning Objectives

1. Create SQL statements to evaluate the model
2. Extract temporal features
3. Perform a feature cross on temporal features

Introduction

In this lab, we utilize feature engineering to improve the prediction of the fare amount for a taxi ride in New York City. We will use BigQuery ML to build a taxifare prediction model, using feature engineering to improve and create a final model.

In this Notebook we set up the environment, create the project dataset, create a feature engineering table, create and evaluate a baseline model, extract temporal features, perform a feature cross on temporal features, and evaluate model performance throughout the process.

Each learning objective will correspond to a **#TODO** in the student lab notebook -- try to complete that notebook first before reviewing this solution notebook.

Set up environment variables and load necessary libraries

In [ ]:
```
# Installing the latest version of the package
!pip install --user google-cloud-bigquery==1.25.0
```

Note: Restart your kernel to use updated packages.

Kindly ignore the deprecation warnings and incompatibility errors related to google-cloud-storage.

In [1]:
```
%%bash
# Exporting the project

export PROJECT=$(gcloud config list project --format "value(core.project)")
echo "Your current GCP Project Name is: "$PROJECT
```

```
Your current GCP Project Name is: munn-sandbox
```

The source dataset

Our dataset is hosted in BigQuery. The taxi fare data is a publically available dataset, meaning anyone with a GCP account has access. Click here to access the dataset.

The Taxi Fare dataset is relatively large at 55 million training rows, but simple to understand, with only six features. The fare_amount is the target, the continuous value we'll train a model to predict.

Create a BigQuery Dataset

A BigQuery dataset is a container for tables, views, and models built with BigQuery ML. Let's create one called **feat_eng** if we have not already done so in an earlier lab. We'll do the same for a GCS bucket for our project too.

In [2]:
```
%%bash

# Create a BigQuery dataset for feat_eng if it doesn't exist
datasetexists=$(bq ls -d | grep -w feat_eng)

if [ -n "$datasetexists" ]; then
    echo -e "BigQuery dataset already exists, let's not recreate it."

else
    echo "Creating BigQuery dataset titled: feat_eng"

    bq --location=US mk --dataset \
        --description 'Taxi Fare' \
        $PROJECT:feat_eng
   echo "\nHere are your current datasets:"
   bq ls
fi
```

```
Creating BigQuery dataset titled: feat_eng
Dataset 'munn-sandbox:feat_eng' successfully created.
\nHere are your current datasets:
    datasetId
 ---------------
  None
  arc_results
  babyweight
  bqml_taxifare
  demo
  demo_sample
  feat_eng
  movielens
  patent_demo
  patents
  stock_market
  taxifare
  test_sample
```

Create the training data table

Since there is already a publicly available dataset, we can simply create the training data table using this raw input data. Note the WHERE clause in the below query: This clause allows us to TRAIN a portion of the data (e.g. one hundred thousand rows versus one million rows), which keeps your query costs down. If you need a refresher on using MOD() for repeatable splits see this post.

Note: The dataset in the create table code below is the one created previously, e.g. "feat_eng". The table name is "featen_training_data". **Run the query to create the table**.

In [3]:
```
%%bigquery

CREATE OR REPLACE TABLE
  feat_eng.featen_training_data AS
SELECT
```

```
  (tolls_amount + fare_amount) AS fare_amount,
  passenger_count*1.0 AS passengers,
  pickup_datetime,
  pickup_longitude AS pickuplon,
  pickup_latitude AS pickuplat,
  dropoff_longitude AS dropofflon,
  dropoff_latitude AS dropofflat
FROM
  `nyc-tlc.yellow.trips`
WHERE
  MOD(ABS(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING))), 10000) = 1
  AND fare_amount >= 2.5
  AND passenger_count > 0
  AND pickup_longitude > -78
  AND pickup_longitude < -70
  AND dropoff_longitude > -78
  AND dropoff_longitude < -70
  AND pickup_latitude > 37
  AND pickup_latitude < 45
  AND dropoff_latitude > 37
  AND dropoff_latitude < 45
```

Out[3]:

Verify table creation

Verify that you created the dataset.

In [4]:

```
%%bigquery

# LIMIT 0 is a free query; this allows us to check that the table exists.
SELECT
*
FROM
  feat_eng.feateng_training_data
LIMIT
  0
```

Out[4]:

fare_amount	passengers	pickup_datetime	pickuplon	pickuplat	dropofflon	dropofflat

Baseline Model: Create the baseline model

Next, you create a linear regression baseline model with no feature engineering. Recall that a model in BigQuery ML represents what an ML system has learned from the training data. A baseline model is a solution to a problem without applying any machine learning techniques.

When creating a BQML model, you must specify the model type (in our case linear regression) and the input label (fare_amount). Note also that we are using the training data table as the data source.

Now we create the SQL statement to create the baseline model.

In [5]:

```
%%bigquery

CREATE OR REPLACE MODEL
```

```
  feat_eng.baseline_model OPTIONS (model_type='linear_reg',
    input_label_cols=['fare_amount']) AS
SELECT
  fare_amount,
  passengers,
  pickup_datetime,
  pickuplon,
  pickuplat,
  dropofflon,
  dropofflat
FROM
  feat_eng.feateng_training_data
```

Out[5]:

Note, the query takes several minutes to complete. After the first iteration is complete, your model (baseline_model) appears in the navigation panel of the BigQuery web UI. Because the query uses a CREATE MODEL statement to create a model, you do not see query results.

You can observe the model as it's being trained by viewing the Model stats tab in the BigQuery web UI. As soon as the first iteration completes, the tab is updated. The stats continue to update as each iteration completes.

Once the training is done, visit the BigQuery Cloud Console and look at the model that has been trained. Then, come back to this notebook.

Evaluate the baseline model

Note that BigQuery automatically split the data we gave it, and trained on only a part of the data and used the rest for evaluation. After creating your model, you evaluate the performance of the regressor using the ML.EVALUATE function. The ML.EVALUATE function evaluates the predicted values against the actual data.

NOTE: The results are also displayed in the BigQuery Cloud Console under the **Evaluation** tab.

Review the learning and eval statistics for the baseline_model.

In [6]:

```
%%bigquery

# Eval statistics on the held out data.
# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
  *,
  SQRT(loss) AS rmse
FROM
  ML.TRAINING_INFO(MODEL feat_eng.baseline_model)
```

Out[6]:

	training_run	iteration	loss	eval_loss	learning_rate	duration_ms	rmse
0	0	0	74.43591	68.880408	None	12426	8.627625

In [7]:

```
%%bigquery
```

```
# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
  *
FROM
  ML.EVALUATE(MODEL feat_eng.baseline_model)
```

Out[7]:

	mean_absolute_error	mean_squared_error	mean_squared_log_error	median_absolute_error	r2_score
0	5.213479	68.880408	0.258098	3.794535	0.226065

NOTE: Because you performed a linear regression, the results include the following columns:

- mean_absolute_error
- mean_squared_error
- mean_squared_log_error
- median_absolute_error
- r2_score
- explained_variance

Resource for an explanation of the Regression Metrics.

Mean squared error (MSE) - Measures the difference between the values our model predicted using the test set and the actual values. You can also think of it as the distance between your regression (best fit) line and the predicted values.

Root mean squared error (RMSE) - The primary evaluation metric for this ML problem is the root mean-squared error. RMSE measures the difference between the predictions of a model, and the observed values. A large RMSE is equivalent to a large average error, so smaller values of RMSE are better. One nice property of RMSE is that the error is given in the units being measured, so you can tell very directly how incorrect the model might be on unseen data.

R2: An important metric in the evaluation results is the R2 score. The R2 score is a statistical measure that determines if the linear regression predictions approximate the actual data. Zero (0) indicates that the model explains none of the variability of the response data around the mean. One (1) indicates that the model explains all the variability of the response data around the mean.

Next, we write a SQL query to take the SQRT() of the mean squared error as your loss metric for evaluation for the benchmark_model.

In [8]:

```
%%bigquery
#TODO 1

# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
  SQRT(mean_squared_error) AS rmse
FROM
  ML.EVALUATE(MODEL feat_eng.baseline_model)
```

Out[8]:

	rmse

	rmse
0	8.299422

Model 1: EXTRACT dayofweek from the pickup_datetime feature.

- As you recall, dayofweek is an enum representing the 7 days of the week. This factory allows the enum to be obtained from the int value. The int value follows the ISO-8601 standard, from 1 (Monday) to 7 (Sunday).
- If you were to extract the dayofweek from pickup_datetime using BigQuery SQL, the datatype returned would be integer.

Next, we create a model titled "model_1" from the benchmark model and extract out the DayofWeek.

In [9]:
```
%%bigquery
#TODO 2

CREATE OR REPLACE MODEL
  feat_eng.model_1 OPTIONS (model_type='linear_reg',
    input_label_cols=['fare_amount']) AS
SELECT
  fare_amount,
  passengers,
  pickup_datetime,
  EXTRACT(DAYOFWEEK
  FROM
    pickup_datetime) AS dayofweek,
  pickuplon,
  pickuplat,
  dropofflon,
  dropofflat
FROM
  feat_eng.feateng_training_data
```

Out[9]: —

Once the training is done, visit the BigQuery Cloud Console and look at the model that has been trained. Then, come back to this notebook.

Next, two distinct SQL statements show the TRAINING and EVALUATION metrics of model_1.

In [104…
```
%%bigquery

# Here, ML.TRAINING_INFO function is used to see information about the training iterati
SELECT
  *,
  SQRT(loss) AS rmse
FROM
  ML.TRAINING_INFO(MODEL feat_eng.model_1)
```

Out[104…

	training_run	iteration	loss	eval_loss	learning_rate	duration_ms	rmse

	training_run	iteration	loss	eval_loss	learning_rate	duration_ms	rmse
0	0	0	72.440724	88.953232	None	17251	8.511212

In [108…

```
%%bigquery

# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
  *
FROM
  ML.EVALUATE(MODEL feat_eng.model_1)
```

Out[108…

	mean_absolute_error	mean_squared_error	mean_squared_log_error	median_absolute_error	r2_score	e
0	5.287076	88.953232	0.260932	3.713439	0.08481	

Here we run a SQL query to take the SQRT() of the mean squared error as your loss metric for evaluation for the benchmark_model.

In [117…

```
%%bigquery

# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
  SQRT(mean_squared_error) AS rmse
FROM
  ML.EVALUATE(MODEL feat_eng.model_1)
```

Out[117…

	rmse
0	9.431502

Model 2: EXTRACT hourofday from the pickup_datetime feature

As you recall, **pickup_datetime** is stored as a TIMESTAMP, where the Timestamp format is retrieved in the standard output format – year-month-day hour:minute:second (e.g. 2016-01-01 23:59:59). Hourofday returns the integer number representing the hour number of the given date.

Hourofday is best thought of as a discrete ordinal variable (and not a categorical feature), as the hours can be ranked (e.g. there is a natural ordering of the values). Hourofday has an added characteristic of being cyclic, since 12am follows 11pm and precedes 1am.

Next, we create a model titled "model_2" and EXTRACT the hourofday from the pickup_datetime feature to improve our model's rmse.

In [11]:

```
%%bigquery
#TODO 3a

CREATE OR REPLACE MODEL
  feat_eng.model_2 OPTIONS (model_type='linear_reg',
    input_label_cols=['fare_amount']) AS
```

```
SELECT
  fare_amount,
  passengers,
  #pickup_datetime,
  EXTRACT(DAYOFWEEK
  FROM
    pickup_datetime) AS dayofweek,
  EXTRACT(HOUR
  FROM
    pickup_datetime) AS hourofday,
  pickuplon,
  pickuplat,
  dropofflon,
  dropofflat
FROM
  `feat_eng.feateng_training_data`
```

Out[11]:

In [12]:
```
%%bigquery

# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
  *
FROM
  ML.EVALUATE(MODEL feat_eng.model_2)
```

Out[12]:

	mean_absolute_error	mean_squared_error	mean_squared_log_error	median_absolute_error	r2_score
0	5.256421	70.709518	0.262399	3.895416	0.236668

In [13]:
```
%%bigquery

# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
  SQRT(mean_squared_error) AS rmse
FROM
  ML.EVALUATE(MODEL feat_eng.model_2)
```

Out[13]:

	rmse
0	8.408895

Model 3: Feature cross dayofweek and hourofday using CONCAT

First, let's allow the model to learn traffic patterns by creating a new feature that combines the time of day and day of week (this is called a feature cross.

Note: BQML by default assumes that numbers are numeric features, and strings are categorical features. We need to convert both the dayofweek and hourofday features to strings because the model (Neural Network) will automatically treat any integer as a numerical value rather than a

categorical value. Thus, if not cast as a string, the dayofweek feature will be interpreted as numeric values (e.g. 1,2,3,4,5,6,7) and hourofday will also be interpreted as numeric values (e.g. the day begins at midnight, 00:00, and the last minute of the day begins at 23:59 and ends at 24:00). As such, there is no way to distinguish the "feature cross" of hourofday and dayofweek "numerically". Casting the dayofweek and hourofday as strings ensures that each element will be treated like a label and will get its own coefficient associated with it.

Create the SQL statement to feature cross the dayofweek and hourofday using the CONCAT function. Name the model "model_3"

In [14]:
```
%%bigquery
#TODO 3b

CREATE OR REPLACE MODEL
  feat_eng.model_3 OPTIONS (model_type='linear_reg',
    input_label_cols=['fare_amount']) AS
SELECT
  fare_amount,
  passengers,
  #pickup_datetime,
  #EXTRACT(DAYOFWEEK FROM pickup_datetime) AS dayofweek,
  #EXTRACT(HOUR FROM pickup_datetime) AS hourofday,
  CONCAT(CAST(EXTRACT(DAYOFWEEK
      FROM
        pickup_datetime) AS STRING), CAST(EXTRACT(HOUR
      FROM
        pickup_datetime) AS STRING)) AS hourofday,
  pickuplon,
  pickuplat,
  dropofflon,
  dropofflat
FROM
  `feat_eng.feateng_training_data`
```

Out[14]: —

In [15]:
```
%%bigquery

# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
  *
FROM
  ML.EVALUATE(MODEL feat_eng.model_3)
```

Out[15]:

	mean_absolute_error	mean_squared_error	mean_squared_log_error	median_absolute_error	r2_score
0	5.265321	69.356068	0.267477	3.887588	0.220371

In [16]:
```
%%bigquery

# Here, ML.EVALUATE function is used to evaluate model metrics
SELECT
```

```
  SQRT(mean_squared_error) AS rmse
FROM
  ML.EVALUATE(MODEL feat_eng.model_3)
```

Out[16]:

	rmse
0	8.328029

Optional: Create a RMSE summary table to evaluate model performance.

Model	Taxi Fare	Description
baseline_model	8.62	Baseline model - no feature engineering
model_1	9.43	EXTRACT dayofweek from the pickup_datetime
model_2	8.40	EXTRACT hourofday from the pickup_datetime
model_3	8.32	FEATURE CROSS hourofday and dayofweek