Машинное обучение в облачном сервисе BigQuery

Что такое BigQuery ML?

BigQuery ML позволяет создавать и выполнять модели машинного обучения в BigQuery, используя стандартные запросы SQL. BigQuery ML демократизирует машинное обучение, позволяя специалистам по SQL создавать модели с использованием существующих инструментов и навыков SQL. BigQuery ML увеличивает скорость разработки за счет устранения необходимости перемещать данные.

Функциональность BigQuery ML доступна при использовании:

- Консоль Google Cloud
- Инструмент bq командной строки
- API REST BigQuery
- Внешний инструмент, такой как блокнот Jupyter или платформа бизнес-аналитики.

Что такое BigQuery ML?

Машинное обучение на больших наборах данных требует обширного программирования и знания фреймворков машинного обучения. Эти требования ограничивают разработку решений очень небольшим кругом людей в каждой компании и исключают аналитиков данных, которые понимают данные, но имеют ограниченные знания в области машинного обучения и программирования.

BigQuery ML позволяет аналитикам данных использовать машинное обучение с помощью существующих инструментов и навыков SQL. Аналитики могут использовать BigQuery ML для построения и оценки моделей машинного обучения в BigQuery. Аналитикам не нужно экспортировать небольшие объемы данных в электронные таблицы или другие приложения или ждать ограниченных ресурсов от группы специалистов по анализу данных.

Поддерживаемые модели в BigQuery ML

Модель в BigQuery ML представляет собой то , что система ML извлекла из обучающих данных. BigQuery ML поддерживает следующие типы моделей:

- Линейная регрессия для прогнозирования; например, продажи товара в определенный день. Метки (Labels) являются действительными (они не могут быть +/- бесконечность или содержать отсутствующие значения NaN).
- Бинарная логистическая регрессия для классификации; например, определение того, совершит ли покупатель покупку. Labels должны иметь только два возможных значения.
- Мультиклассовая логистическая регрессия для решения задачи классификации. Эти модели могут использоваться для прогнозирования нескольких возможных значений, например, является ли выход «низким значением», «средним значением» или «высоким значением». Labels могут иметь до 50 уникальных значений. В BigQuery ML при обучении мультиклассовой логистической регрессии используется полиномиальный классификатор с cross-entropy loss function.
- K-means кластеризация для сегментации данных; например, определение клиентских сегментов. K-means - это метод обучения без учителя, поэтому для обучения модели не требуются метки или разделение данных для обучения или оценки.

Поддерживаемые модели в BigQuery ML

- Matrix Factorization для создания систем рекомендаций по продуктам. Можно создавать рекомендации по продуктам, используя исторические данные о поведении клиентов, транзакциях и рейтингах продуктов, а затем использовать эти рекомендации для персонализированного взаимодействия с клиентами.
- Time series ряды для выполнения прогнозов временных рядов. Можно использовать эту функцию для создания миллионов моделей временных рядов и использовать их для прогнозирования. Модель автоматически обрабатывает аномалии, сезонность и праздники в данных.
- Boosted Tree для создания моделей классификации и регрессии на основе XGBoost.
- Deep Neural Network (DNN) для создания глубоких нейронных сетей на основе TensorFlow для моделей классификации и регрессии .
- AutoML Tables для создания лучших в своем классе моделей без проектирования элементов или выбора модели. AutoML Tables выполняет поиск среди множества архитектур моделей, чтобы выбрать лучшую модель.
- TensorFlow model importing. Эта функция позволяет создавать модели BigQuery ML из ранее обученных моделей TensorFlow, а затем выполнять прогнозирование в BigQuery ML.

Преимущества BigQuery ML

- BigQuery ML демократизирует использование машинного обучения, давая возможность аналитикам данных, основным пользователям хранилищ данных, создавать и запускать модели с использованием существующих инструментов бизнес-аналитики. Прогнозная аналитика может направлять процесс принятия бизнес-решений в масштабах всей организации.
- Нет необходимости программировать решение машинного обучения с использованием Python или Java. Модели обучаются и доступны в BigQuery с помощью SQL - языка, известного аналитикам данных.
- BigQuery ML увеличивает скорость разработки моделей и инноваций, устраняя необходимость экспортировать данные из хранилища данных. Вместо этого BigQuery ML использует ML для обработки данных. Необходимость экспорта и переформатирования данных имеет следующие недостатки:
  - Повышает сложность, потому что требуется несколько инструментов.
  - Снижает скорость, поскольку перемещение и форматирование больших объемов данных для платформ машинного обучения на основе Python занимает больше времени, чем обучение модели в BigQuery.
  - Требуется несколько шагов для экспорта данных из хранилища, что ограничивает возможность экспериментировать с вашими данными.
  - Может быть предотвращено с помощью юридических ограничений.

Оператор CREATE MODEL для обобщенных линейных моделей

Чтобы создать модель линейной регрессии или логистической регрессии в BigQuery, используйте оператор CREATE MODEL BigQuery ML с типами моделей LINEAR_REG или LOGISTIC_REG.

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL}
model_name
[OPTIONS(MODEL_TYPE = { 'LINEAR_REG' | 'LOGISTIC_REG' },
    INPUT_LABEL_COLS = string_array,
    OPTIMIZE_STRATEGY = { 'AUTO_STRATEGY' | 'BATCH_GRADIENT_DESCENT' | 'NORMAL_EQUATION' },
    L1_REG = float64_value,
    L2_REG = float64_value,
    MAX_ITERATIONS = int64_value,
    LEARN_RATE_STRATEGY = { 'LINE_SEARCH' | 'CONSTANT' },
    LEARN_RATE = float64_value,
    EARLY_STOP = { TRUE | FALSE },
    MIN_REL_PROGRESS = float64_value,
    DATA_SPLIT_METHOD = { 'AUTO_SPLIT' | 'RANDOM' | 'CUSTOM' | 'SEQ' | 'NO_SPLIT' },
    DATA_SPLIT_EVAL_FRACTION = float64_value,
    DATA_SPLIT_COL = string_value,
    LS_INIT_LEARN_RATE = float64_value,
    WARM_START = { TRUE | FALSE },
    AUTO_CLASS_WEIGHTS = { TRUE | FALSE },
    CLASS_WEIGHTS = struct_array
)];
```

Обучение модели линейной регрессии

В следующем примере создается и обучается модель линейной регрессии. Скорость обучения установлена на 0,15, регуляризация L1 установлена на 1, а максимальное количество итераций обучения установлено на 5.

```
CREATE MODEL
  `mydataset.mymodel`
OPTIONS
  ( MODEL_TYPE='LINEAR_REG',
    LS_INIT_LEARN_RATE=0.15,
    L1_REG=1,
    MAX_ITERATIONS=5 ) AS
SELECT
  column1,
  column2,
  column3,
  label
FROM `mydataset.mytable`
WHERE
  column4 < 10
```

Обучение модели линейной регрессии с последовательным разделением данных

В этом примере вы создаете модель линейной регрессии с последовательным разделением данных. Доля разделения составляет 0,3, а в качестве основы для разделения используется столбец timestamp.

```
CREATE MODEL
  `mydataset.mymodel`
OPTIONS
  ( MODEL_TYPE='LINEAR_REG',
    LS_INIT_LEARN_RATE=0.15,
    L1_REG=1,
    MAX_ITERATIONS=5,
    DATA_SPLIT_METHOD='SEQ',
    DATA_SPLIT_EVAL_FRACTION=0.3,
    DATA_SPLIT_COL='timestamp' ) AS
SELECT
  column1,
  column2,
  column3,
  timestamp,
  label
FROM `mydataset.mytable`
WHERE
  column4 < 10
```

Данные обучения используются для обучения модели. Данные оценки используются, чтобы избежать переобучения за счет ранней остановки.

Обучение многоклассовой модели логистической регрессии с автоматически рассчитанными весами

В этом примере вы создаете многоклассовую модель логистической регрессии с помощью auto_class_weightsпараметра.

```
CREATE MODEL
  `mydataset.mymodel`
OPTIONS
  ( MODEL_TYPE='LOGISTIC_REG',
    AUTO_CLASS_WEIGHTS=TRUE ) AS
SELECT
  *
FROM `mydataset.mytable`
```

Обучение многоклассовой модели логистической регрессии с заданными весами

В этом примере вы создаете многоклассовую модель логистической регрессии с помощью class_weightsпараметра. Label columns являются label1, label2 и label3.

```
CREATE MODEL
  `mydataset.mymodel`
OPTIONS
  ( MODEL_TYPE='LOGISTIC_REG',
    CLASS_WEIGHTS=[('label1', 0.5), ('label2', 0.3), ('label3', 0.2)]) AS
SELECT
  *
FROM `mydataset.mytable`
```

Оператор CREATE MODEL для моделей K-means

Чтобы создать модель K-means в BigQuery, используйте оператор CREATE MODEL Big Query ML с типом модели KMEANS.

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL}
model_name
[OPTIONS(MODEL_TYPE = { 'KMEANS' },
    NUM_CLUSTERS = int64_value,
    KMEANS_INIT_METHOD = { 'RANDOM' | 'KMEANS++' | 'CUSTOM' },
    KMEANS_INIT_COL = string_value,
    DISTANCE_TYPE = { 'EUCLIDEAN' | 'COSINE' },
    STANDARDIZE_FEATURES = { TRUE | FALSE },
    MAX_ITERATIONS = int64_value,
    EARLY_STOP = { TRUE | FALSE },
    MIN_REL_PROGRESS = float64_value,
    WARM_START = { TRUE | FALSE }
)];
```

Обучение модели k-means с методом случайной инициализации кластера

В этом примере создается модель k-средних с тремя кластерами с использованием метода случайной инициализации кластера.

```
CREATE MODEL
  `mydataset.mymodel`
OPTIONS
  ( MODEL_TYPE='KMEANS',
    NUM_CLUSTERS=3,
    KMEANS_INIT_METHOD='RANDOM') AS
SELECT
  *
FROM `mydataset.mytable`
```

Обучение модели k-means с помощью настраиваемого метода инициализации кластера

В этом примере создается модель k-средних с четырьмя кластерами с использованием метода инициализации настраиваемого кластера. init_col определяет столбец типа, BOOL который содержит значения, определяющие, является ли данная строка начальным центроидом. Этот столбец должен содержать только три строки со значением TRUE.

```
CREATE MODEL
  `mydataset.mymodel`
OPTIONS
  ( MODEL_TYPE='KMEANS',
    NUM_CLUSTERS=3,
    KMEANS_INIT_METHOD='CUSTOM',
    KMEANS_INIT_COL='init_col') AS
SELECT
  init_col,
  features
FROM `mydataset.mytable`
```

Оператор CREATE MODEL для моделей Boosted Tree с использованием XGBoost

Чтобы создать модель Boosted Tree в BigQuery, используйте CREATE MODELоператор BigQuery ML с типами модели BOOSTED_TREE_CLASSIFIER или BOOSTED_TREE_REGRESSOR. Модель обучается с использованием библиотеки XGBoost .

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL} model_name
[OPTIONS(MODEL_TYPE = { 'BOOSTED_TREE_CLASSIFIER' | 'BOOSTED_TREE_REGRESSOR' },
         BOOSTER_TYPE = {'GBTREE' | 'DART'},
         NUM_PARALLEL_TREE = int64_value,
         DART_NORMALIZE_TYPE = {'TREE' | 'FOREST'},
         TREE_METHOD = {'AUTO' | 'EXACT' | 'APPROX' | 'HIST'},
         MIN_TREE_CHILD_WEIGHT = int64_value,
         COLSAMPLE_BYTREE = float64_value,
         COLSAMPLE_BYLEVEL = float64_value,
         COLSAMPLE_BYNODE = float64_value,
         MIN_SPLIT_LOSS = float64_value,
         MAX_TREE_DEPTH = int64_value,
         SUBSAMPLE = float64_value,
         AUTO_CLASS_WEIGHTS = { TRUE | FALSE },
         CLASS_WEIGHTS = struct_array,
         L1_REG = float64_value,
         L2_REG = float64_value,
         EARLY_STOP = { TRUE | FALSE },
         LEARN_RATE = float64_value,
         INPUT_LABEL_COLS = string_array,
         MAX_ITERATIONS = int64_value,
         MIN_REL_PROGRESS = float64_value,
         DATA_SPLIT_METHOD = { 'AUTO_SPLIT' | 'RANDOM' | 'CUSTOM' | 'SEQ' | 'NO_SPLIT' },
         DATA_SPLIT_EVAL_FRACTION = float64_value,
         DATA_SPLIT_COL = string_value
)];
```

В следующем примере выполняется обучение модели классификатора Boosted Tree ' mydataset.mytable’ с использованием label column 'mylabel'.

```
CREATE MODEL project_id:mydataset.mymodel
OPTIONS(MODEL_TYPE='BOOSTED_TREE_CLASSIFIER',
      BOOSTER_TYPE = 'GBTREE',
      NUM_PARALLEL_TREE = 1,
      MAX_ITERATIONS = 50,
      TREE_METHOD = 'HIST',
      EARLY_STOP = FALSE,
      SUBSAMPLE = 0.85,
      INPUT_LABEL_COLS = ['mylabel'])
AS SELECT * FROM project_id:mydataset.mytable;
```

- NUM_PARALLEL_TREE = int64_value

Количество параллельных деревьев, построенных на каждой итерации. Значение по умолчанию - 1. Чтобы обучить усиленный случайный лес, установите это значение больше 1.

- HIST (метод обучения) рекомендуется для больших наборов данных, чтобы достичь более высокой скорости обучения и снизить потребление ресурсов.
- SUBSAMPLE = float64_value . Соотношение подвыборки обучающих примеров. Установка этого значения на 0,5 означает, что при обучении случайным образом выбирается половина обучающих данных перед выращиванием деревьев, что предотвращает переобучение.

Оператор CREATE MODEL для моделей глубокой нейронной сети (DNN)

Чтобы создать модель глубокой нейронной сети в BigQuery, используйте CREATE MODELоператор BigQuery ML с типами модели DNN_CLASSIFIER или DNN_REGRESSOR. Эти модели построены с использованием TensorFlow estimators.

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL} model_name
[OPTIONS(MODEL_TYPE = { 'DNN_CLASSIFIER' | 'DNN_REGRESSOR' },
         ACTIVATION_FN = { 'RELU' | 'RELU6' | 'CRELU' | 'ELU' | 'SELU' | 'SIGMOID' | 'TANH' },
         AUTO_CLASS_WEIGHTS = { TRUE | FALSE },
         BATCH_SIZE = int64_value,
         CLASS_WEIGHTS = struct_array,
         DROPOUT = float64_value,
         EARLY_STOP = { TRUE | FALSE },
         HIDDEN_UNITS = int_array,
         LEARN_RATE = float64_value,
         INPUT_LABEL_COLS = string_array,
         MAX_ITERATIONS = int64_value,
         MIN_REL_PROGRESS = float64_value,
         OPTIMIZER = { 'ADAGRAD' | 'ADAM' | 'FTRL' | 'RMSPROP' | 'SGD' },
         WARM_START = { TRUE | FALSE },
         DATA_SPLIT_METHOD = { 'AUTO_SPLIT' | 'RANDOM' | 'CUSTOM' | 'SEQ' | 'NO_SPLIT' },
         DATA_SPLIT_EVAL_FRACTION = float64_value,
         DATA_SPLIT_COL = string_value
)];
```

В следующем примере выполняется обучение модели классификатора DNN 'mytable'c 'mylabel’ с использованием label column.

```
CREATE MODEL project_id:mydataset.mymodel
OPTIONS(MODEL_TYPE='DNN_CLASSIFIER',
        ACTIVATION_FN = 'RELU',
        BATCH_SIZE = 16,
        DROPOUT = 0.1,
        EARLY_STOP = FALSE,
        HIDDEN_UNITS = [128, 128, 128],
        INPUT_LABEL_COLS = ['mylabel'],
        LEARN_RATE=0.001,
        MAX_ITERATIONS = 50,
        OPTIMIZER = 'ADAGRAD')
AS SELECT * FROM project_id:mydataset.mytable;
```

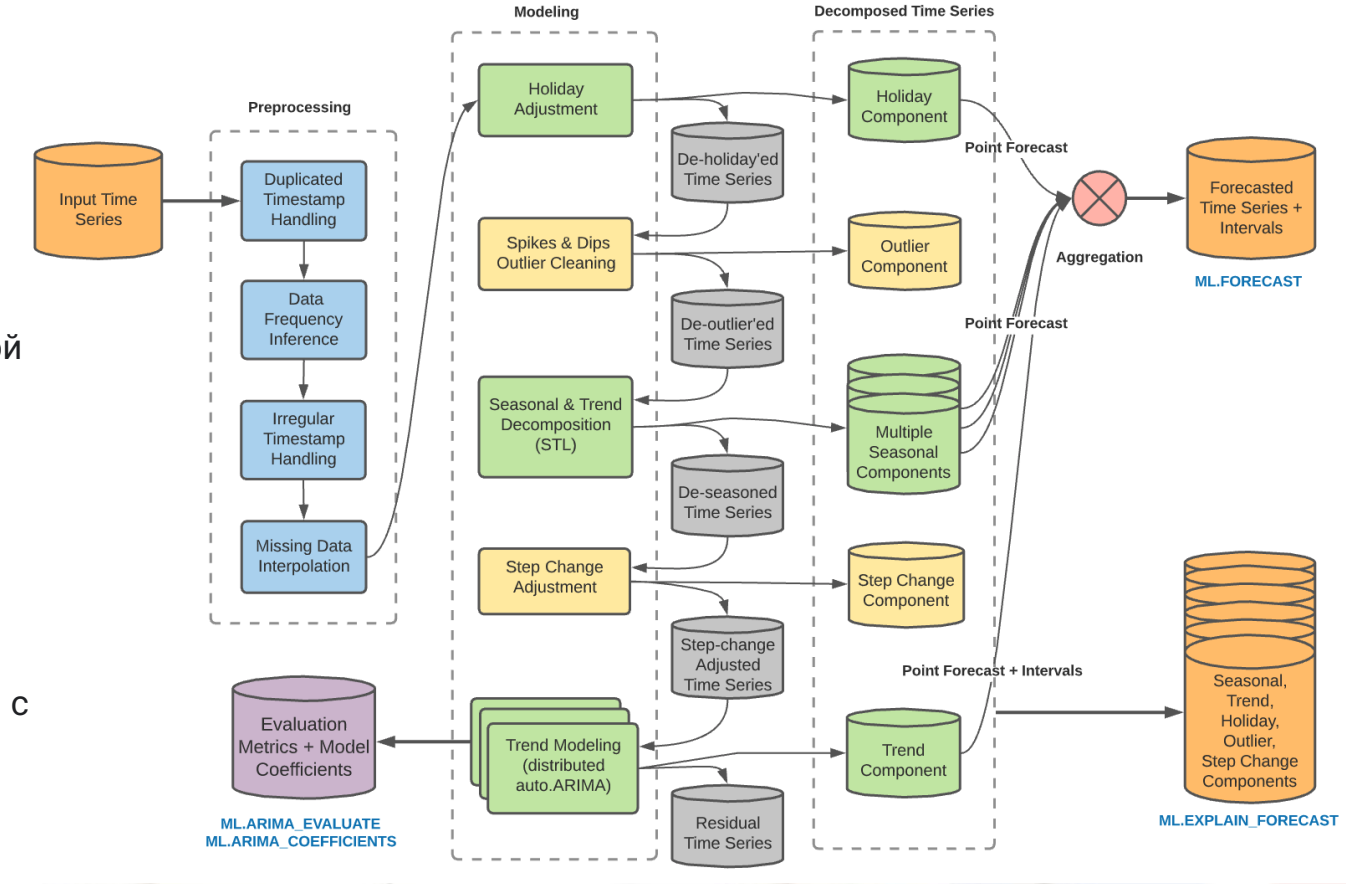
- размер мини-пакета выборок, которые передаются в нейронную сеть = 16.
- частоту отсева единиц в нейронной сети = 0.1.
- Следует ли останавливать обучение после первой итерации, в которой улучшение относительных потерь меньше значения, указанного для MIN_REL_PROGRESS.
- архитектура из 3 скрытых слоев с 128, 128 и 128 нейронами соответственно
- Начальная скорость обучения для обучения = 0.001
- Число эпох обучения = 50 (эпоха - один проход всех обучающих данных.)
- OPTIMIZER = 'ADAGRAD’) - Оптимизатор градиентного спуска со скоростью обучения, зависящей от прошлых градиентов, в каждом измерении.

Оператор CREATE MODEL для моделей временных рядов

Чтобы создать модели временных рядов в BigQuery, используйте CREATE MODELоператор BigQuery ML и укажите MODEL_TYPEзначение be 'ARIMA_PLUS'.

Конвейер моделирования для временных рядов BigQuery ML включает следующие функции:

- Вывод о частоте данных временного ряда.
- Обращаться с нерегулярными временными интервалами.
- Обрабатывать повторяющиеся отметки времени, беря среднее значение.
- Интерполировать отсутствующие данные с помощью локальной линейной интерполяции.
- Обнаружение и очистка выбросов с пиками и падениями.
- Обнаружение и корректировка резких скачков (уровней).
- Обнаружение и настройка эффекта праздничных дней.
- Обнаружение нескольких сезонных закономерностей в одном временном ряду с помощью разложения по сезонам и трендам с использованием лесса (STL) и экстраполяция сезонности с помощью двойного экспоненциального сглаживания (ETS) .
- Обнаружение и моделирование тенденции с помощью модели ARIMA и алгоритма auto.ARIMA для автоматической настройки гиперпараметров. В auto.ARIMA параллельно обучаются и оцениваются десятки моделей-кандидатов. Лучшая модель имеет самый низкий информационный критерий Акаике (AIC) .



{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL} *model_name* OPTIONS(MODEL_TYPE = 'ARIMA_PLUS'
[, TIME_SERIES_TIMESTAMP_COL = ***string_value***]
[, TIME_SERIES_DATA_COL = ***string_value***]
[, TIME_SERIES_ID_COL = { ***string_value*** | ***string_array*** }]
[, HORIZON = ***int64_value***]
[, AUTO_ARIMA = { TRUE | FALSE }]
[, AUTO_ARIMA_MAX_ORDER = ***int64_value***]
[, NON_SEASONAL_ORDER = ***(int64_value, int64_value, int64_value)***]
[, DATA_FREQUENCY = { 'AUTO_FREQUENCY' | 'PER_MINUTE' | 'HOURLY' | 'DAILY' | 'WEEKLY' | 'MONTHLY' | 'QUARTERLY' | 'YEARLY' }]
[, INCLUDE_DRIFT = { TRUE | FALSE }]
[, HOLIDAY_REGION = { 'GLOBAL' | 'NA' | 'JAPAC' | 'EMEA' | 'LAC' | 'AE' | ... }]
[, CLEAN_SPIKES_AND_DIPS = { TRUE | FALSE }]
[, ADJUST_STEP_CHANGES = { TRUE | FALSE }]
[, DECOMPOSE_TIME_SERIES = { TRUE | FALSE }])
AS *query_statement*

Авторегрессионное интегрированное скользящее среднее (ARIMA) является обобщением модели **авторегрессионного скользящего среднего**. Эти модели используются при работе с временными рядами для более глубокого понимания данных или предсказания будущих точек ряда.
В модели ARIMA(p,d,q), где p,d и q — целые неотрицательные числа, характеризующие порядок для частей модели (соответственно авторегрессионной, интегрированной и скользящего среднего).

AUTO_ARIMA: следует ли при обучении использовать auto.ARIMA или нет. Если true, обучение автоматически найдет лучший несезонный порядок (т.е. кортеж **p, d, q**) и решит, включать ли член линейного дрейфа, когда d равно 1. Если false, пользователь должен указать non_seasonal_order в поле запрос. При прогнозировании нескольких временных рядов одновременно алгоритм auto.ARIMA должен использоваться для каждого временного ряда, поэтому для этого параметра не должно быть установлено значение false.

Обучение модели временных рядов для прогнозирования одного временного ряда

```
CREATE MODEL
`project_id.mydataset.mymodel`
 OPTIONS(MODEL_TYPE='ARIMA_PLUS',
        time_series_timestamp_col='date',
        time_series_data_col='transaction') AS
SELECT
  date,
  transaction
FROM
  `mydataset.mytable`
```

Одновременное обучение нескольких моделей временных рядов для нескольких временных рядов

В этом примере создается несколько моделей временных рядов, по одной для каждого входного временного ряда.

```
CREATE MODEL `project_id.mydataset.mymodel`
 OPTIONS(MODEL_TYPE='ARIMA_PLUS',
        time_series_timestamp_col='date',
        time_series_data_col='transaction',
        time_series_id_col='company_name') AS
SELECT
  date,
  transaction,
  company_name
FROM
  `mydataset.mytable`
```

Оператор CREATE MODEL для матричной факторизации

Чтобы создать модель матричной факторизации в BigQuery, используйте CREATE MODELоператор BigQuery ML и укажите MODEL_TYPEзначение be 'MATRIX_FACTORIZATION'.

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL}
model_name
OPTIONS(MODEL_TYPE = 'MATRIX_FACTORIZATION'
        FEEDBACK_TYPE = {'EXPLICIT' | 'IMPLICIT'},
        NUM_FACTORS = int64_value,
        USER_COL = string_value,
        ITEM_COL = string_value,
        RATING_COL = string_value,
        WALS_ALPHA = float64_value,
        L2_REG = float64_value,
        MAX_ITERATIONS = int64_value,
        EARLY_STOP = { TRUE | FALSE },
        MIN_REL_PROGRESS = float64_value,
        DATA_SPLIT_METHOD = { 'AUTO_SPLIT' | 'RANDOM' | 'CUSTOM' | 'SEQ' | 'NO_SPLIT' },
        DATA_SPLIT_EVAL_FRACTION = float64_value,
        DATA_SPLIT_COL = string_value)
AS query_statement
```

Обучение модели матричной факторизации с явной обратной связью

```
CREATE MODEL `project_id.mydataset.mymodel`
 OPTIONS(MODEL_TYPE='MATRIX_FACTORIZATION') AS
SELECT
  user,
  item,
  rating
FROM `mydataset.mytable`
```

Обучение модели матричной факторизации с неявной обратной связью

```
CREATE MODEL
`project_id.mydataset.mymodel`
 OPTIONS(MODEL_TYPE='MATRIX_FACT
ORIZATION',
        FEEDBACK_TYPE='IMPLICIT') AS
SELECT
  user,
  item,
  rating
FROM `mydataset.mytable`
```

Тип обратной связи определяет алгоритм, который используется во время обучения.
Существует два типа оценок (отзывы пользователей): **'EXPLICIT'** и **'IMPLICIT'**. Используйте желаемый тип обратной связи в параметрах создания модели в зависимости от варианта использования. Если пользователь явно поставил оценку (например, от 1 до 5) такому элементу, как рекомендации фильмов, укажите FEEDBACK_TYPE='EXPLICIT'. Это обучит модель с использованием алгоритма альтернативных наименьших квадратов.
Большинство проблем с рекомендациями по продукту не имеют явной обратной связи с пользователем. Вместо этого значение рейтинга должно быть искусственно построено на основе взаимодействия пользователя с элементом (например, кликов, просмотров страниц и покупок). В этой ситуации укажите FEEDBACK_TYPE='IMPLICIT'. Это обучит модель с использованием алгоритма взвешенных альтернативных наименьших квадратов.

Оператор CREATE MODEL для обучения моделей AutoML Tables

AutoML Tables позволяет автоматически создавать современные модели машинного обучения на основе структурированных (табличных) данных со значительно увеличенной скоростью.

Ограничения:

- Входные данные для таблиц AutoML должны быть от 1000 до 100 миллионов строк и менее 100 ГБ.
- Шифрование не поддерживается.
- Модели не отображаются в пользовательском интерфейсе AutoML Tables и недоступны для пакетных или интерактивных прогнозов в таблицах AutoML.

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL}
model_name
OPTIONS(MODEL_TYPE = { 'AUTOML_REGRESSOR' | 'AUTOML_CLASSIFIER' },
        BUDGET_HOURS = float64_value,
        OPTIMIZATION_OBJECTIVE = float64_value,
        INPUT_LABEL_COLS = string_array,
        DATA_SPLIT_COL = string)
[AS query_statement];
```

Следующий пример создает модель, названную mymodel в mydataset в проекте по умолчанию. Он использует общедоступные данные о поездках на такси nyc-tlc.yellow.trips, доступные в BigQuery. Выполнение задания занимает около 3 часов, включая обучение, сжатие модели, временное перемещение данных (в AutoML) и задачи настройки.

```
CREATE OR REPLACE MODEL project_id.mydataset.mymodel
      OPTIONS(model_type='AUTOML_REGRESSOR',
              input_label_cols=['fare_amount'],
              budget_hours=1.0)
AS SELECT
  (tolls_amount + fare_amount) AS fare_amount,
  pickup_longitude,
  pickup_latitude,
  dropoff_longitude,
  dropoff_latitude,
  passenger_count
FROM `nyc-tlc.yellow.trips`
WHERE ABS(MOD(FARM_FINGERPRINT(CAST(pickup_datetime AS STRING)),
100000)) = 1
AND
  trip_distance > 0
  AND fare_amount >= 2.5 AND fare_amount <= 100.0
  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
  AND passenger_count > 0
```

В настоящее время BigQuery ML использует значения по умолчанию для параметров обучения AutoML Tables, включая автоматическое разделение данных и функцию оптимизации по умолчанию.
Для регрессионной модели тип должен быть «AUTOML_REGRESSOR», а тип столбца метки должен быть числовым. Для модели классификации тип должен быть «AUTOML_CLASSIFIER», а столбец метки может быть строковым или числовым.
Бюджет обучения для обучения AutoML Tables, указан в часах. По умолчанию 1.0 и должен быть от 1.0 до 72.0. После обучения моделей AutoML Tables BigQuery ML сжимает модель, чтобы убедиться, что она достаточно мала для импорта, что может занять до 50% времени обучения. Время сжатия модели не входит во время обучения.
Целевая функцию оптимизации, которая будет использоваться для обучения AutoML Tables: для регрессии допустимые значения: MINIMIZE_RMSE (по умолчанию), MINIMIZE_MAE и MINIMIZE_RMSLE. Для двоичной классификации: MAXIMIZE_AU_ROC (по умолчанию), MINIMIZE_LOG_LOSS, MAXIMIZE_AU_PRC. Для мультиклассовой классификации единственное допустимое значение: MINIMIZE_LOG_LOSS.

Оператор CREATE MODEL для импорта моделей TensorFlow

Создает новую модель BigQuery ML в указанном наборе данных. Для моделей TensorFlow BigQuery ML импортирует существующую модель TensorFlow и преобразует ее в модель BigQuery ML.

Ограничения (наиболее важные):

- Модель TensorFlow должна уже существовать, прежде чем ее можно будет импортировать в BigQuery ML.
- Модели должны храниться в облачном хранилище.
- Модели замораживаются во время создания модели.
- Модели TensorFlow должны быть в формате SavedModel .
- Размер модели ограничен 250 МБ.
- Поддерживаются только основные операции TensorFlow

```
{CREATE MODEL | CREATE MODEL IF NOT EXISTS | CREATE OR REPLACE MODEL}
model_name
[OPTIONS(MODEL_TYPE = 'TENSORFLOW', MODEL_PATH = string_value)];
```

В следующем примере модель TensorFlow импортируется в BigQuery ML как модель BigQuery ML. В примере предполагается, что существует существующая модель TensorFlow, расположенная по адресу gs://bucket/path/to/saved_model/*.

```
CREATE MODEL project_id.mydataset.mymodel
 OPTIONS(MODEL_TYPE='TENSORFLOW',
         MODEL_PATH="gs://bucket/path/to/saved_model/*")
```