Repeatable splitting

In this notebook, we will explore the impact of different ways of creating machine learning datasets.

Repeatability is important in machine learning. If you do the same thing now and 5 minutes from now and get different answers, then it makes experimentation difficult. In other words, you will find it difficult to gauge whether a change you made has resulted in an improvement or not.

In [1]:

```
!sudo chown -R jupyter:jupyter /home/jupyter/training-data-analyst
```

In [2]:

```
!pip install --user google-cloud-bigquery==1.25.0
```

```
Collecting google-cloud-bigquery==1.25.0
  Downloading google_cloud_bigquery-1.25.0-py2.py3-none-any.whl (169 kB)
     |████████████████████████████████| 169 kB 5.2 MB/s eta 0:00:01
Requirement already satisfied: six<2.0.0dev,>=1.13.0 in /opt/conda/lib/pyth
on3.7/site-packages (from google-cloud-bigquery==1.25.0) (1.16.0)
Requirement already satisfied: google-auth<2.0dev,>=1.9.0 in /opt/conda/lib
/python3.7/site-packages (from google-cloud-bigquery==1.25.0) (1.34.0)
Collecting google-resumable-media<0.6dev,>=0.5.0
  Downloading google_resumable_media-0.5.1-py2.py3-none-any.whl (38 kB)
Requirement already satisfied: google-cloud-core<2.0dev,>=1.1.0 in /opt/con
da/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0) (1.7.2)
Requirement already satisfied: google-api-core<2.0dev,>=1.15.0 in /opt/cond
a/lib/python3.7/site-packages (from google-cloud-bigquery==1.25.0) (1.31.1)
Requirement already satisfied: protobuf>=3.6.0 in /opt/conda/lib/python3.7/
site-packages (from google-cloud-bigquery==1.25.0) (3.16.0)
Requirement already satisfied: packaging>=14.3 in /opt/conda/lib/python3.7/
site-packages (from google-api-core<2.0dev,>=1.15.0->google-cloud-bigquery=
=1.25.0) (21.0)
Requirement already satisfied: googleapis-common-protos<2.0dev,>=1.6.0 in /
opt/conda/lib/python3.7/site-packages (from google-api-core<2.0dev,>=1.15.0
->google-cloud-bigquery==1.25.0) (1.53.0)
Requirement already satisfied: requests<3.0.0dev,>=2.18.0 in /opt/conda/lib
/python3.7/site-packages (from google-api-core<2.0dev,>=1.15.0->google-clou
d-bigquery==1.25.0) (2.25.1)
Requirement already satisfied: pytz in /opt/conda/lib/python3.7/site-packag
es (from google-api-core<2.0dev,>=1.15.0->google-cloud-bigquery==1.25.0) (2
021.1)
Requirement already satisfied: setuptools>=40.3.0 in /opt/conda/lib/python
3.7/site-packages (from google-api-core<2.0dev,>=1.15.0->google-cloud-bigqu
ery==1.25.0) (49.6.0.post20210108)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /opt/conda/lib/pyth
on3.7/site-packages (from google-auth<2.0dev,>=1.9.0->google-cloud-bigquery
==1.25.0) (0.2.7)
Requirement already satisfied: cachetools<5.0,>=2.0.0 in /opt/conda/lib/pyt
hon3.7/site-packages (from google-auth<2.0dev,>=1.9.0->google-cloud-bigquer
y==1.25.0) (4.2.2)
Requirement already satisfied: rsa<5,>=3.1.4 in /opt/conda/lib/python3.7/si
te-packages (from google-auth<2.0dev,>=1.9.0->google-cloud-bigquery==1.25.
0) (4.7.2)
Requirement already satisfied: pyparsing>=2.0.2 in /opt/conda/lib/python3.7
/site-packages (from packaging>=14.3->google-api-core<2.0dev,>=1.15.0->goog
le-cloud-bigquery==1.25.0) (2.4.7)
```

```
Requirement already satisfied: pyasn1<0.5.0,>=0.4.6 in /opt/conda/lib/pytho
n3.7/site-packages (from pyasn1-modules>=0.2.1->google-auth<2.0dev,>=1.9.0-
>google-cloud-bigquery==1.25.0) (0.4.8)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python
3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-api-core<2.0dev,
>=1.15.0->google-cloud-bigquery==1.25.0) (2021.5.30)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/sit
e-packages (from requests<3.0.0dev,>=2.18.0->google-api-core<2.0dev,>=1.15.
0->google-cloud-bigquery==1.25.0) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/lib/python3.
7/site-packages (from requests<3.0.0dev,>=2.18.0->google-api-core<2.0dev,>=
1.15.0->google-cloud-bigquery==1.25.0) (4.0.0)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/pyth
on3.7/site-packages (from requests<3.0.0dev,>=2.18.0->google-api-core<2.0de
v,>=1.15.0->google-cloud-bigquery==1.25.0) (1.26.6)
Installing collected packages: google-resumable-media, google-cloud-bigquer
y
ERROR: pip's dependency resolver does not currently take into account all t
he packages that are installed. This behaviour is the source of the followi
ng dependency conflicts.
tfx-bsl 1.2.0 requires absl-py<0.13,>=0.9, but you have absl-py 0.13.0 whic
h is incompatible.
tfx-bsl 1.2.0 requires google-api-python-client<2,>=1.7.11, but you have go
ogle-api-python-client 2.15.0 which is incompatible.
tfx-bsl 1.2.0 requires google-cloud-bigquery<2.21,>=1.28.0, but you have go
ogle-cloud-bigquery 1.25.0 which is incompatible.
tfx-bsl 1.2.0 requires pyarrow<3,>=1, but you have pyarrow 5.0.0 which is i
ncompatible.
tensorflow-transform 1.2.0 requires absl-py<0.13,>=0.9, but you have absl-p
y 0.13.0 which is incompatible.
tensorflow-transform 1.2.0 requires google-cloud-bigquery<2.21,>=1.28.0, bu
t you have google-cloud-bigquery 1.25.0 which is incompatible.
tensorflow-transform 1.2.0 requires pyarrow<3,>=1, but you have pyarrow 5.
0.0 which is incompatible.
google-cloud-storage 1.41.1 requires google-resumable-media<3.0dev,>=1.3.0;
python_version >= "3.6", but you have google-resumable-media 0.5.1 which is
incompatible.
Successfully installed google-cloud-bigquery-1.25.0 google-resumable-media-
0.5.1
```

Restart the kernel before proceeding further (On the Notebook menu - Kernel - Restart Kernel).

In [3]:

```python
from google.cloud import bigquery
```

Create a simple machine learning model

The dataset that we will use is a BigQuery public dataset of airline arrival data. Click on the link, and look at the column names. Switch to the Details tab to verify that the number of records is 70 million, and then switch to the Preview tab to look at a few rows.

We want to predict the arrival delay of an airline based on the departure delay. The model that we will use is a zero-bias linear model: $$ delay_{arrival} = \alpha * delay_{departure} $$

To train the model is to estimate a good value for α.

One approach to estimate alpha is to use this formula: $$ \alpha = \frac{\sum delay_{departure} delay_{arrival} }{ \sum delay_{departure}^2 } $$ Because we'd like to capture the idea that this relationship is different for flights from New York to Los Angeles vs. flights from Austin to Indianapolis (shorter flight, less busy airports), we'd compute a different $alpha$ for each airport-pair. For simplicity, we'll do this model only for flights between Denver and Los Angeles.

Naive random split (not repeatable)

In [4]:

```
compute_alpha = """
#standardSQL
SELECT
    SAFE_DIVIDE(
    SUM(arrival_delay * departure_delay),
    SUM(departure_delay * departure_delay)) AS alpha
FROM
(
    SELECT
        RAND() AS splitfield,
        arrival_delay,
        departure_delay
    FROM
        `bigquery-samples.airline_ontime_data.flights`
    WHERE
        departure_airport = 'DEN'
        AND arrival_airport = 'LAX'
)
WHERE
    splitfield < 0.8
"""
```

In [5]:

```
results = bigquery.Client().query(compute_alpha).to_dataframe()
alpha = results['alpha'][0]
print(alpha)
```

```
0.9746759869400119
```

What is wrong with calculating RMSE on the training and test data as follows?

In [6]:

```
compute_rmse = """
#standardSQL
SELECT
    dataset,
    SQRT(
        AVG(
            (arrival_delay - ALPHA * departure_delay) *
            (arrival_delay - ALPHA * departure_delay)
        )
    ) AS rmse,
    COUNT(arrival_delay) AS num_flights
FROM (
    SELECT
        IF (RAND() < 0.8, 'train', 'eval') AS dataset,
        arrival_delay,
        departure_delay
    FROM
        `bigquery-samples.airline_ontime_data.flights`
    WHERE
        departure_airport = 'DEN'
        AND arrival_airport = 'LAX' )
GROUP BY
    dataset
"""
bigquery.Client().query(compute_rmse.replace('ALPHA', str(alpha))).to_data
```

Out[6]:

	dataset	rmse	num_flights
0	eval	13.060312	16042
1	train	13.089737	63647

Hint:

- Are you really getting the same training data in the compute_rmse query as in the compute_alpha query?
- Do you get the same answers each time you rerun the compute_alpha and compute_rmse blocks?

How do we correctly train and evaluate?

Here's the right way to compute the RMSE using the actual training and held-out (evaluation) data. Note how much harder this feels.

Although the calculations are now correct, the experiment is still not repeatable.

Try running it several times; do you get the same answer?

In [7]:

```
train_and_eval_rand = """
#standardSQL
WITH
    alldata AS (
        SELECT
            IF (RAND() < 0.8, 'train', 'eval') AS dataset,
            arrival_delay,
            departure_delay
        FROM
            `bigquery-samples.airline_ontime_data.flights`
        WHERE
        departure_airport = 'DEN'
        AND arrival_airport = 'LAX' ),
    training AS (
        SELECT
            SAFE_DIVIDE(
                SUM(arrival_delay * departure_delay),
                SUM(departure_delay * departure_delay)) AS alpha
        FROM
            alldata
        WHERE
            dataset = 'train' )

SELECT
    MAX(alpha) AS alpha,
    dataset,
    SQRT(
        AVG(
            (arrival_delay - alpha * departure_delay) *
            (arrival_delay - alpha * departure_delay)
        )
    ) AS rmse,
    COUNT(arrival_delay) AS num_flights
FROM
    alldata,
    training
GROUP BY
    dataset
"""
```

In [8]:

```
bigquery.Client().query(train_and_eval_rand).to_dataframe()
```

Out[8]:

	alpha	dataset	rmse	num_flights
0	0.975873	train	13.023342	63689
1	0.975873	eval	13.321639	16000

Using HASH of date to split the data

Let's split by date and train.

In [9]:

```
compute_alpha = """
#standardSQL
SELECT
    SAFE_DIVIDE(
        SUM(arrival_delay * departure_delay),
        SUM(departure_delay * departure_delay)) AS alpha
FROM
    `bigquery-samples.airline_ontime_data.flights`
WHERE
    departure_airport = 'DEN'
    AND arrival_airport = 'LAX'
    AND ABS(MOD(FARM_FINGERPRINT(date), 10)) < 8
"""
results = bigquery.Client().query(compute_alpha).to_dataframe()
alpha = results['alpha'][0]
print(alpha)
```

```
0.9758039143620403
```

We can now use the alpha to compute RMSE. Because the alpha value is repeatable, we don't need to worry that the alpha in the compute_rmse will be different from the alpha computed in the compute_alpha.

In [10]:

```
compute_rmse = """
#standardSQL
SELECT
    IF(ABS(MOD(FARM_FINGERPRINT(date), 10)) < 8, 'train', 'eval') AS datase
    SQRT(
        AVG(
            (arrival_delay - ALPHA * departure_delay) *
            (arrival_delay - ALPHA * departure_delay)
        )
    ) AS rmse,
    COUNT(arrival_delay) AS num_flights
FROM
    `bigquery-samples.airline_ontime_data.flights`
WHERE
    departure_airport = 'DEN'
    AND arrival_airport = 'LAX'
GROUP BY
    dataset
"""
print(bigquery.Client().query(compute_rmse.replace('ALPHA', str(alpha))).t
```

```
  dataset       rmse  num_flights
0    eval  12.764685        15671
1   train  13.160712        64018
```

Note also that the RMSE on the evaluation dataset more from the RMSE on the training dataset when we do the split correctly. This should be expected; in the RAND() case, there was leakage between training and evaluation datasets, because there is high correlation between flights on the same day.

This is one of the biggest dangers with doing machine learning splits the wrong way -- **you will develop a false sense of confidence in how good your model is!**